2019

Efficient state estimation via inference on a probabilistic graphical model

Luke David Myers
Iowa State University

Efficient state estimation via inference on a probabilistic graphical model

by

Luke David Myers

A thesis submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Electrical Engineering (Electric Power and Energy Systems)

Program of Study Committee:
Daji Qiao, Major Professor
Zhenqiang Gong
Venkataramana Ajjarapu

The student author, whose presentation of the scholarship herein was approved by the program of study committee, is solely responsible for the content of this thesis. The Graduate College will ensure this thesis is globally accessible and will not permit alterations after a degree is conferred.

Iowa State University

Ames, Iowa

2019

DEDICATION

I would like to dedicate this thesis to my father, who has always supported and encouraged me both in my educational endeavors as well as in every other area of my life.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

Page

ACKNOWLEDGMENTS

I want to express my gratitude and thanks to those who have assisted me in conducting the research for this thesis during my graduate studies. First of all, I would like to thank my Major Professor Dr. Daji Qiao for all of his guidance, motivation, and support throughout this research. I would also like to thank fellow graduate student Binghui Wang for his work in developing the probabilistic graphical model and indispensable contributions to the research behind this thesis. Additionally, I want to thank my committee members Dr. Zhenqiang Gong and Dr. Venkataramana Ajjarapu, as well as Dr. Guan, Grant Johnson, and Benjamin Blakely, for their help and oversight throughout this research. Finally, I would like to thank Argonne National Laboratory and Ames Laboratory for their financial support of this research during my time as a graduate student.

ABSTRACT

This thesis presents a unique and efficient solver to the state estimation (SE) problem for the power grid, based on probabilistic graphical models (PGMs). SE is a method of estimating the varying state values of voltage magnitude and phase at every bus within a power grid based on meter measurements. However, existing SE solvers are notorious for their computational inefficiency to calculate the matrix inverse, and hence slow convergence to produce the final state estimates. The proposed PGM-based solver estimates the state values from a different perspective. Instead of calculating the matrix inverse directly, it models the power grid as a PGM, and then assigns potentials to nodes and edges of the PGM, based on the physical constraints of the power grid. This way, the original SE problem is transformed into an equivalent probabilistic inference problem on the PGM, for which two efficient algorithms are proposed based on Gaussian belief propagation (GBP). The equivalence between the proposed PGM-based solver and existing SE solvers is shown in terms of state estimates, and it is experimentally demonstrated that this new method converges much faster than existing solvers.

CHAPTER 1. INTRODUCTION

1.1 Background

Power grid is a complex energy delivery system comprised of a large number of individual components and geographically dispersed communication systems. State estimation (SE) is a method of estimating the varying state values of voltage magnitude and phase at every bus of a power grid. This is done by taking a series of meter measurements of key parameters across the network and relaying those measurements to the control center. These meter measurements are then used to estimate the voltage states based upon their physical relations.

SE can be classified as DC SE and AC SE. AC SE estimates state values by solving a system of (complex) nonlinear equations (considering both real and reactive power), while DC SE utilizes a linear approximation of the nonlinear real power flow equations in AC SE to estimate the state values (while neglecting reactive power) [1, 2]. Existing solvers to both DC SE and AC SE involve the calculation of the inverse of a matrix. Direct matrix inverse methods (e.g., Gaussian elimination algorithm) are computationally infeasible if the scale of the power grid (and hence the size of the matrix) is large. Iterative methods, such as the Jacobi iterative method and the Gauss–Seidel iterative method, can avoid the calculation of matrix inverse. However, one major issue with the iterative methods [3] is that they may require a very large number of iterations and converge slowly or even cannot converge. To address this issue, we propose to solve SE from a different probabilistic graphical model (PGM) perspective, which converges much faster and thus is a much more efficient solution to the SE problem.

PGM [4] is a technique that offers a compact graph-based representation of joint probability distributions, and exploits conditional independencies among the random variables. PGM has been widely used in many research areas such as machine learning, signal processing, computer vision, and data mining. To the best of our knowledge, we are the first ones to leverage PGM to improve the efficiency of state estimation.

The key idea of our proposed PGM solver to the SE problem is to represent a power grid as a PGM, and then transform the original SE problem into an equivalent probabilistic inference problem on the PGM. By doing so, the physical constraints and relations between components in the power grid are considered explicitly in the PGM and during the computation. Moreover, by utilizing the Gaussian belief propagation (GBP) algorithm to perform the probabilistic inference, each node in the PGM takes messages from all of its neighbor nodes at each iteration, thus progressing more toward the final estimates at each iteration. Hence, our proposed PGM solver requires much fewer iterations to converge.

Given the fast converging property of the proposed PGM solver, it has many potential applications. For example, it can be used, together with the AC flow model, as an efficient and practical defense to the well-known and well-studied false data injection attack (FDIA) [1] against a power grid.

1.2 Our Contributions

The contributions of this thesis include:

- We propose a novel framework to perform SE based on probabilistic graphical models (PGMs).
- We design two Gaussian belief propagation (GBP) algorithms to perform the probabilistic inference on our PGM.
- We prove and demonstrate the equivalence between existing SE solvers and the proposed PGM solver in terms of state estimates.
- We demonstrate experimentally that the proposed PGM solver requires a much shorter running time than standard SE solvers, particularly when the power grid is large.
- As an example application of the proposed PGM solver, we show that it can effectively and quickly detect the FDIA, when used together with the AC flow model.
- All experiments are conducted with the MATPOWER simulator [5]. Codes are developed to perform DC SE and the proposed PGM solver, and to produce the meter measurements, for a variety of power grid models.

1.3 Organization

The rest of this thesis is organized as follows. In Chapter 2, we provide a brief overview of SE on both DC and AC flow models, existing methods to solve SE, and FDIA against DC SE. In Chapter 3, we show the background of probabilistic graphical model (PGM) and the inference algorithm. Chapter 4 details our proposed PGM and designs two inference algorithms to solve SE. Chapter 5 evaluates our proposed method on a series of experiments that compare PGM with standard SE. Chapter 6 explores the applicability of AC PGM as a means of defense against FDIAs. Chapter 7 evaluates the resilience of PGM to FDIAs through a number of experiments. Finally, the thesis is concluded in Chapter 8. Appendices A and B include MATLAB code that was generated to perform some of the experiments.

1.4 My Own Contributions

Since I am not the sole individual responsible for the research for, work on, and composition of all the material presented in this thesis, it is necessary that I identify my own personal contributions before moving forward. I am largely responsible for most of the background research and work on standard DC SE, AC SE, and FDIA reflected in Chapters 1, 2, and 6. I also conducted all of the experiments detailed in Chapters 5 and 7 (with the exception of running the PGM tests in Chapter 5). The MATLAB code included in Appendices A and B are also contributions for which I am solely responsible. In addition to this, I am responsible for having edited all of the material presented in this thesis.

Chapters 3–4, which detail our probabilistic graphical model and its application to the power grid for state estimation, is almost exclusively the work of Binghui Wang, who is the only other main contributor to the content contained in this thesis. Where appropriate, I use singular personal pronouns throughout this thesis to refer to myself as the main contributor. The use of plural pronouns outside of Chapters 3–4 are used to indicate joint contribution on the part of both Binghui Wang and myself. Other contributors to the work behind this thesis are Dr. Daji Qiao and Dr. Zhenqiang Gong, who oversaw my research process and provided me with invaluable guidance and direction. It should also be noted that we have submitted a paper for publication in *IEEE Transactions on Smart Grid* that includes much of the material presented in this thesis.

CHAPTER 2. STATE ESTIMATION

2.1 State Estimation On DC And AC Flow Models

State estimation (SE) is a method of estimating state variables of a power grid so that operators can monitor them from the control center. The state variables of interest are the voltage magnitude, $|u|$, and phase, θ_u, at every bus in the network. In order to perform SE for a given power system, a set of meter measurements need to be taken of various components across the network and reported to the control center. These meter measurements can then be used along with topology information of the grid to estimate the state variables. Table 2.1 summarizes the terms and notations used in Chapters 2 to 4.

When performing SE, one of the generator buses in the power system is selected to be the *slack* or *reference bus*, at which the voltage phase is defined as $\theta_u = 0$ radians. This leaves $2N-1$ unknown states to be estimated, where N is the number of buses in the system. The task of SE is to estimate these $2N-1$ states, given the meter measurements and the network topology of the power system. At least $2N-1$ meter measurements are required to estimate $2N-1$ states.

Let m represent the number of states ($m = 2N-1$) and n represent the number of meter measurements ($n \geqslant m$) utilized to perform state estimation. Moreover, we denote the m state variables as $\mathbf{x} = [x_1, x_2, \ldots, x_m]^T$, n meter measurements as $\mathbf{z} = [z_1, z_1, \cdots, z_n]^T$, and the meter measurement errors as $\mathbf{e} = [e_1, e_1, \cdots, e_n]^T$. Then, the relation between $\mathbf{x}$, $\mathbf{z}$, and $\mathbf{e}$ is encoded in the following system of equations:

$$\mathbf{z} = \mathbf{h}(\mathbf{x}) + \mathbf{e}, \tag{2.1}$$

where $\mathbf{h}(\mathbf{x}) = [h_1(x_1, \cdots, x_m), h_2(x_1, \cdots, x_m), \cdots, h_n(x_1, \cdots, x_m)]^T$ is an n-dim vector function that establishes dependencies between $\mathbf{x}$ and $\mathbf{z}$. Each meter measurement error e_i is assumed to be Gaussian-distributed with zero mean and variance $\boldsymbol{\sigma^2}$, i.e., $e_i \sim \mathcal{N}(0, \sigma_i^2)$.

Table 2.1 Terms and notations used in Chapters 2 to 4

Symbol	Definition
SE	State estimation
AC SE	Nonlinear state estimation
DC SE	Linear state estimation
$\|u\|$	Voltage magnitude
θ_u	Voltage phase
N	Number of buses
m	Number of states
n	Number of measurements
$\mathbf{x}$	State variables
$\mathbf{z}$	Meter measurements
$\mathbf{e}$	Meter measurement errors
$J(\mathbf{x})$	Objective function
Λ	Matrix of meter measurement error variances
$\mathbf{H}$	Jacobian matrix
$\tilde{\mathbf{x}}$	Augmented state variables
$\tilde{\mathbf{z}}$	Augmented meter measurements
$\tilde{\mathbf{H}}$	Augmented Jacobian matrix
$\phi_{\tilde{v}_i}(\tilde{x}_i)$	Node potential
$\psi_{\tilde{v}_i,\tilde{v}_j}(\tilde{x}_i,\tilde{x}_j)$	Edge potential

The goal of SE is to estimate state variables that best fit Eq. (2.1). Specifically, SE aims to minimize the following objective function:

$$J(\mathbf{x}) = \frac{1}{2}\sum_{i=1}^{n}\left(\frac{z_i - h_i(\mathbf{x})}{\sigma_i}\right)^2 = \frac{1}{2}(\mathbf{z}-\mathbf{h}(\mathbf{x}))^T\Lambda(\mathbf{z}-\mathbf{h}(\mathbf{x})), \tag{2.2}$$

where Λ is a diagonal matrix whose elements are reciprocals of the variances of meter measurement errors, i.e., $\Lambda = \text{diag}(\sigma_1^{-2}, \sigma_2^{-2}, \cdots, \sigma_n^{-2})$. Moreover, a matrix $\mathbf{H}(\mathbf{x})$ that contains the derivative of $\mathbf{h}(\mathbf{x})$ with respect to each state variable x_i is needed. Specifically, it is an $n \times m$ Jacobian matrix with full column rank, as follows:

$$\mathbf{H}(\mathbf{x}) = \begin{bmatrix} \frac{\partial h_1(\mathbf{x})}{\partial x_1} & \frac{\partial h_1(\mathbf{x})}{\partial x_2} & \cdots & \frac{\partial h_1(\mathbf{x})}{\partial x_m} \\ \frac{\partial h_2(\mathbf{x})}{\partial x_1} & \frac{\partial h_2(\mathbf{x})}{\partial x_2} & \cdots & \frac{\partial h_n(\mathbf{x})}{\partial x_m} \\ \cdots & & & \\ \frac{\partial h_n(\mathbf{x})}{\partial x_1} & \frac{\partial h_n(\mathbf{x})}{\partial x_2} & \cdots & \frac{\partial h_n(\mathbf{x})}{\partial x_m} \end{bmatrix}. \tag{2.3}$$

If $\mathbf{h}(\mathbf{x})$ relates state variables to meter measurements *linearly*, i.e., $\mathbf{h}(\mathbf{x}) = \mathbf{H}\mathbf{x}$ with a constant matrix $\mathbf{H}$, the above SE problem is called DC SE. In contrast, if $\mathbf{h}(\mathbf{x})$ relates state variables to meter measurements *nonlinearly*, the above SE problem is called AC SE.

2.2 Existing SE Solvers

Existing solutions to SE are based on the fact that, at the minimum of $J(\mathbf{x})$, the first-order optimality conditions have to be satisfied. That is,

$$\mathbf{g}(\mathbf{x}) = \frac{\partial J(\mathbf{x})}{\partial \mathbf{x}} = -\mathbf{H}^T(\mathbf{x})\Lambda(\mathbf{z} - \mathbf{h}(\mathbf{x})) = 0. \tag{2.4}$$

In DC SE, the closed-form solution to Eq. (2.4) can be derived, which is in the following form:

$$\mathbf{x} = (\mathbf{H}^T\Lambda\mathbf{H})^{-1}\mathbf{H}^T\Lambda\mathbf{z}, \tag{2.5}$$

In AC SE, since $\mathbf{h}(\mathbf{x})$ is a nonlinear function of state variables $\mathbf{x}$, a closed-form solution to Eq. (2.4) cannot be obtained. In practice, Newton-type methods are often used to iteratively estimate the state variables. Specifically, the Newton method performs a Taylor series expansion of $\mathbf{g}(\mathbf{x})$ around $\mathbf{x}^{(k)}$, and neglects the higher-order terms. That is,

$$\mathbf{g}(\mathbf{x}^{(k)} + \Delta\mathbf{x}^{(k)}) = \mathbf{g}(\mathbf{x}^{(k)}) + \mathbf{G}(\mathbf{x}^{(k)})\Delta\mathbf{x}^{(k)} = \mathbf{0}, \tag{2.6}$$

where

$$\mathbf{G}(\mathbf{x}^{(k)}) = \frac{\partial g(\mathbf{x}^{(k)})}{\partial \mathbf{x}} = \mathbf{H}(\mathbf{x}^{(k)})^T\Lambda\mathbf{H}(\mathbf{x}^{(k)}). \tag{2.7}$$

Then, with an initial setting $\mathbf{x}^{(0)}$, the Newton method iteratively updates $\mathbf{x}^{(k)}$ for $k = 1, 2, \cdots$, as follows:

$$\begin{aligned}\Delta\mathbf{x}^{(k)} &= -(\mathbf{G}(\mathbf{x}^{(k)}))^{-1}\mathbf{g}(\mathbf{x}^{(k)}) \\ &= \left(\mathbf{H}(\mathbf{x}^{(k)})^T\Lambda\mathbf{H}(\mathbf{x}^{(k)})\right)^{-1}\mathbf{H}(\mathbf{x}^{(k)})^T\Lambda(\mathbf{z} - \mathbf{h}(\mathbf{x}^{(k)})); \qquad (2.8)\\ \mathbf{x}^{(k+1)} &= \mathbf{x}^{(k)} + \Delta\mathbf{x}^{(k)}. \qquad (2.9)\end{aligned}$$

Finally, the iterative process is terminated when the ℓ_∞-norm of $\Delta\mathbf{x}^{(k)}$ is less than a predefined threshold ε, or when the process reaches a predefined maximal number of iterations T.

2.3 A 3-bus Example

I will now use a simple 3-bus power grid model (which is taken from problem 9.1 in [6]) to illustrate the SE process. The 3-bus model is shown in Fig. 2.1. The available information for the 3-bus model includes the voltage magnitude at each bus (1 *per unit* or $1pu$), the real power measurements on each of the three transmission lines connecting the buses: $\mathbf{z} = [0.6pu, 0.04pu, 0.405pu]^T$, the standard deviation of the meter measurements: $\sigma = [0.02pu, 0.01pu, 0.002pu]^T$, and the susceptance on each of the transmission lines: $b_{12} = 5pu$, $b_{10} = 2.5pu$, $b_{20} = 4pu$.

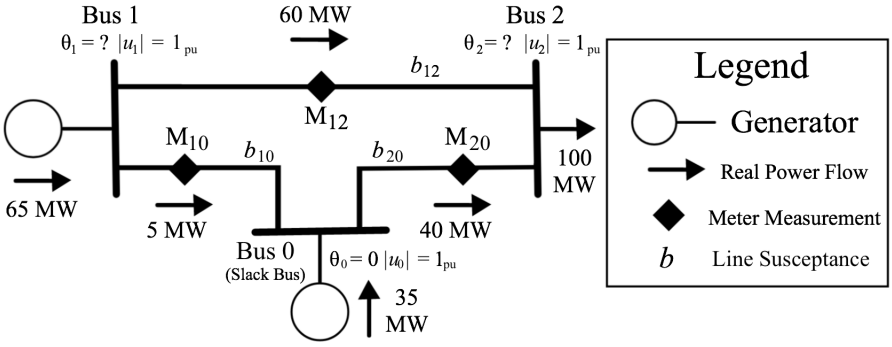


Figure 2.1 An example 3-bus power grid with 3 meters (M_{12}, M_{10}, M_{20}) measuring the real power flow across the transmission lines. The line susceptances are $b_{12} = 5pu$, $b_{10} = 2.5pu$, and $b_{20} = 4pu$, respectively, with a 100 MVA base. Two states, $|u|$ and θ, are present at each of the three buses. In this example, all the voltage magnitudes are assumed to be $1pu$. With Bus 0 being the slack bus, i.e., $\theta_0 = 0$, we have θ_1 and θ_2 as the unknown states to be estimated.

The unknown states to be estimated are θ_1 (the voltage phase at Bus 1) and θ_2 (the voltage phase at Bus 2). For AC SE, we use the nonlinear real power flow equations, as follows:

$$h_1(x) = P_{12} = b_{12}\sin(\theta_1 - \theta_2) = 5\sin(\theta_1 - \theta_2);$$
$$h_2(x) = P_{10} = b_{10}\sin(\theta_1 - \theta_0) = 2.5\sin\theta_1;$$
$$h_3(x) = P_{20} = b_{20}\sin(\theta_0 - \theta_2) = -4\sin\theta_2.$$

Since $\mathbf{h}(\mathbf{x})$ is made up of nonlinear equations, the Jacobian is a function of the unknown states:

$$\mathbf{H}(\mathbf{x}) = \begin{bmatrix} 5\cos(\theta_1 - \theta_2) & -5\cos(\theta_1 - \theta_2) \\ 2.5\cos(\theta_1) & 0 \\ 0 & -4\cos(\theta_2) \end{bmatrix}.$$

For DC SE, we use the linear approximations of the above real power flow equations, as follows:

$$h_1(x) \approx 5\theta_1 - 5\theta_2;$$
$$h_2(x) \approx 2.5\theta_1;$$
$$h_3(x) \approx -4\theta_2.$$

As a result, the Jacobian becomes a constant matrix:

$$\mathbf{H}(\mathbf{x}) = \mathbf{H} = \begin{bmatrix} 5 & -5 \\ 2.5 & 0 \\ 0 & -4 \end{bmatrix}.$$

With $\Lambda = \text{diag}(0.02^{-2}, 0.01^{-2}, 0.002^{-2})$, we have everything needed to obtain the state estimates, $\hat{\mathbf{x}}$. Performing AC SE with Eq. (2.8), we obtain the estimates of θ_1 and θ_2 as 0.0174 and -0.1014 radians, respectively. Performing DC SE with Eq. (2.5), we obtain the estimates of θ_1 and θ_2 as 0.0174 and -0.1013 radians, respectively. Since this example network is a very simple 3-bus system, the results of AC SE and DC SE are very close to each other.

2.4 Discussion

Note that both Eq. (2.5) and Eq. (2.8) involve the calculation of the inverse of a matrix. As noted previously, direct matrix inverse methods are computationally infeasible if the matrix size is large, while iterative methods may require a very large number of iterations and converge slowly or even cannot converge. To address this issue, we propose a novel method to solve the SE problem from a different probabilistic graphical model (PGM) perspective, which we will show converges much faster than existing solvers.

CHAPTER 3. PROBABILISTIC GRAPHICAL MODEL

Probabilistic graphical model (PGM) [4] combines probability theory and graph theory. It offers a compact graph-based representation of joint probability distributions, and exploits conditional independencies among the random variables. Conditional independence can alleviate the computational burden. PGM has been widely used in various research areas such as computer vision [7, 8], speech processing [9], time-series and sequential data modeling [10], cognitive science [11], bioinformatics [12], signal processing [13], communications and error-correcting coding theory [14], and in the area of artificial intelligence in general.

The framework of PGM could provide general techniques for inference (e.g., sum-product message-passing algorithm, also known as *belief propagation*) [15] and learning [4]. Two popular representations of graphical models are *Markov networks* based on undirected graphs (also known as Markov random fields), and *Bayesian networks* based on directed graphs (also known as belief networks). In this work, we mainly focus on Markov networks because the graphical representation of a power grid is undirected.

3.1 Markov Networks, Node Potential, And Edge Potential

Markov networks are represented by undirected graphs. Given an undirected graph $\tilde{G} = (\tilde{V}, \tilde{E})$, where $\tilde{V}$ is a set of nodes and $\tilde{E}$ is a set of edges. Each node $\tilde{v}_i \in \tilde{V}$ is associated with a random variable $\tilde{x}_i$. Markov networks specify a *node potential* $\phi_{\tilde{v}_i}(\tilde{x}_i)$ (also known as "evidence") for each node $\tilde{v}_i$, and specify an *edge potential* $\psi_{\tilde{v}_i,\tilde{v}_j}(\tilde{x}_i,\tilde{x}_j)$ (also known as "compatibility functions") for each edge $(\tilde{v}_i,\tilde{v}_j) \in \tilde{E}$. Then, Markov networks define a joint probability distribution over all random variables as follows:

$$p(\tilde{\mathbf{x}}) = p(x_1, \cdots, x_{|\tilde{V}|}) = \frac{1}{Z} \prod_{\tilde{v}_i \in \tilde{V}} \phi_{\tilde{v}_i}(\tilde{x}_i) \prod_{(\tilde{v}_i,\tilde{v}_j) \in \tilde{E}} \psi_{\tilde{v}_i,\tilde{v}_j}(\tilde{x}_i,\tilde{x}_j), \tag{3.1}$$

where $Z = \sum_{\tilde{x}_i} \prod_{\tilde{v}_i \in \tilde{V}} \phi_{\tilde{v}_i}(\tilde{x}_i) \prod_{(\tilde{v}_i,\tilde{v}_j) \in \tilde{E}} \psi_{\tilde{v}_i,\tilde{v}_j}(\tilde{x}_i,\tilde{x}_j)$ is called the *partition function*, which is used to normalize the probabilities.

Nevertheless, performing inference for each variable, i.e., calculating the marginal distribution of each variable, involves a summation of exponential terms for discrete variables, or an integration for continuous variables. This renders the direct computation intractable for a large number of variables. Fortunately, the underlying graph structure, and thereby the introduced factorization of the joint probability, can be exploited via belief propagation [15].

3.2 Inference Via Belief Propagation

Belief propagation (BP), also called sum-product, is a message-passing algorithm to perform inference on graphical models. BP is originally derived for exact inference in trees [15], and then generalized to a general graph even with loops. Specifically, BP functions by passing real-valued messages across edges in the graph and is comprised of two computational rules: *sum-product* rule and *product* rule.

Sum-product rule: Messages $m_{\tilde{v}_i \to \tilde{v}_j}(\tilde{x}_j)$ are sent on each edge $(\tilde{v}_i, \tilde{v}_j) \in \tilde{E}$ from $\tilde{v}_i$ to $\tilde{v}_j$. If $\tilde{x}_j$ is a discrete random variable, the messages are updated as follows:

$$m_{\tilde{v}_i \to \tilde{v}_j}(\tilde{x}_j) = \sum_{\tilde{x}_i} \psi_{\tilde{v}_i, \tilde{v}_j}(\tilde{x}_i, \tilde{x}_j) \phi_{\tilde{v}_i}(\tilde{x}_i) \prod_{v_k \in N(\tilde{v}_i) \setminus \tilde{v}_j} m_{v_k \to \tilde{v}_i}(\tilde{x}_i), \tag{3.2}$$

where $N(\tilde{v}_i)$ denotes the neighbor nodes of $\tilde{v}_i$, and $N(\tilde{v}_i) \setminus \tilde{v}_j$ means node $\tilde{v}_j$ is excluded from the neighbor nodes. If $\tilde{x}_j$ is a continuous random variable, the messages are updated as follows:

$$m_{\tilde{v}_i \to \tilde{v}_j}(\tilde{x}_j) = \int_{\tilde{x}_i} \psi_{\tilde{v}_i, \tilde{v}_j}(\tilde{x}_i, \tilde{x}_j) \phi_{\tilde{v}_i}(\tilde{x}_i) \prod_{v_k \in N(\tilde{v}_i) \setminus \tilde{v}_j} m_{v_k \to \tilde{v}_i}(\tilde{x}_i) dx_i. \tag{3.3}$$

Fig. 3.1 visualizes the sum-product rule.

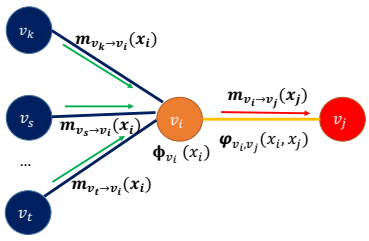


Figure 3.1 Sum-product rule

Product rule: If the Markov network does not have a loop, the sum-product algorithm is *exact* [15], meaning that, if we use the sum-product algorithm, the marginals (also known as *beliefs*) $p(\tilde{x}_i)$ obtained via the following product rule are guaranteed to converge to the true marginals:

$$p(\tilde{x}_i) = \phi_{\tilde{v}_i}(\tilde{x}_i) \prod_{k \in N(\tilde{v}_i)} m_{v_k \to \tilde{v}_i}(\tilde{x}_i). \tag{3.4}$$

Fig. 3.2 visualizes the product rule.

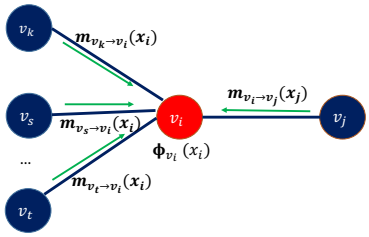


Figure 3.2 Product rule

CHAPTER 4. STATE ESTIMATION: A PGM PERSPECTIVE

In this chapter, we describe our proposed approach to perform SE from a unique perspective, based on PGMs. Comparing with existing SE solvers (shown in Fig. 4.1), our approach consists of the following three steps (shown in Fig. 4.2).

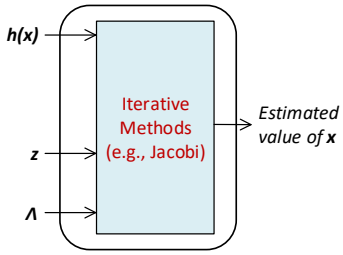


Figure 4.1 Flowchart of existing SE solvers

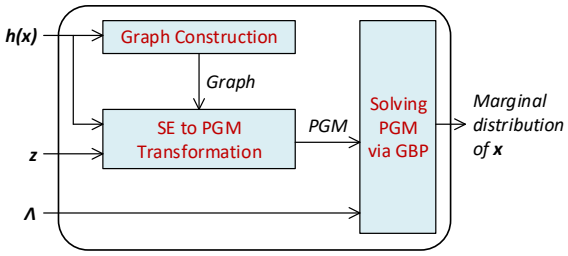


Figure 4.2 Flowchart of our proposed PGM method

First, we model the power grid as an undirected graph (Section 4.1). Then, we contruct a PGM and assign potentials to nodes and edges of the PGM, based on the physical constraints of the power grid, which, in turn, transforms the original state estimation problem into an equivalent inference problem on the PGM (Section 4.2). Finally, we apply Gaussian Belief Propagation (GBP) and design two GBP algorithms to infer the state variables of our PGM, which produce the final state estimation results (Section 4.3).

4.1 Modeling A Power Grid As A Graph

We represent the power grid as an undirected graph $G = (V, E)$. Each node $v_i \in V$ represents a bus. Each edge in E represents a power transmission line between two buses. The weight of each edge is the impedance of the corresponding power transmission line. For each node $v_i \in V$, a node v_j ($\neq v_i$) is a neighbor of v_i if and only if there is an edge between them. The power grid state at a node is determined by (1) the states of its neighbors, and (2) the impedance of the transmission lines between the node and its neighbors. Therefore, we can constrain each node in the power grid graph with local equations, which describe the relation between its state and its neighbors' states. Subsequently, we can use a set of local equations to represent the constraints for the entire power grid, which are given in Eq. (2.1).

Throughout this section, we will use the same example 3-bus power system introduced in Section 2.3 and shown in Fig. 2.1, to illustrate our proposed PGM method, step by step. In the first step, this 3-bus power system is represented by the graph in Fig. 4.3a.

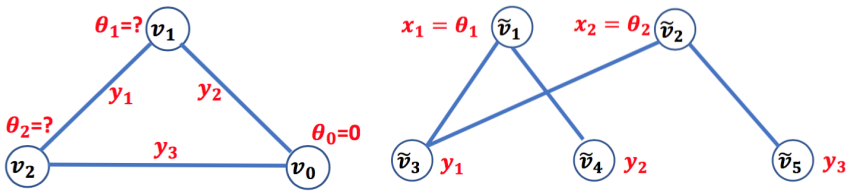


Figure 4.3 (a) Graph representation of the example 3-bus power system. (b) The corresponding PGM graph.

4.2 Transforming SE To PGM Problem

The second step of our approach is to transform SE to an inference problem on a PGM. Next, we first describe how to do it for DC SE, and then we generalize it to AC SE.

4.2.1 Transforming DC SE To Inference Problem On A PGM

The goal of DC SE is to solve Eq. (2.5) to produce an estimation of state variables $\mathbf{x}$. When leveraging PGM for probabilistic inference, we require $\mathbf{H}$ to be a square matrix. However, $\mathbf{H}$ usually is not square in power grids. To address this issue, we adopt the following two steps:

- **Step I**: We transform Eq. (2.5) to a more concise form. Specifically, we denote $\hat{\mathbf{H}} = \Lambda^{1/2}\mathbf{H}$ and $\hat{\mathbf{z}} = \Lambda^{1/2}\mathbf{z}$, where $\Lambda^{1/2} = \text{diag}(\sigma_1^{-1}, \sigma_2^{-1}, \cdots, \sigma_n^{-1})$. Then, Eq. (2.5) becomes:

$$\mathbf{x} = \left(\hat{\mathbf{H}}^T\hat{\mathbf{H}}\right)^{-1}\hat{\mathbf{H}}^T\hat{\mathbf{z}}. \tag{4.1}$$

- **Step II**: We construct an augmented $(m+n)\times(m+n)$ matrix $\tilde{\mathbf{H}}$ for $\hat{\mathbf{H}}$ as:

$$\tilde{\mathbf{H}} = \begin{bmatrix} \mathbf{I}_{m\times m} & \hat{\mathbf{H}}^T \\ \hat{\mathbf{H}} & -\lambda\mathbf{I}_{n\times n} \end{bmatrix}, \tag{4.2}$$

  where $\mathbf{I}_{m\times m}$ and $\mathbf{I}_{n\times n}$ are an $m\times m$ identity matrix and an $n\times n$ identity matrix, respectively. λ is a small constant. Similarly, we define an augmented $(m+n)$-dim vector of state variables $\tilde{\mathbf{x}} = [\mathbf{x};\mathbf{y}]$, where $\mathbf{y}$ is an n-dim auxiliary variable vector, and an augmented $(m+n)$-dim measurement vector $\tilde{\mathbf{z}} = [\mathbf{0}_m;\hat{\mathbf{z}}]$, where $\mathbf{0}_m$ is an m-dim zero vector.

Then, we have the following theorem.

Theorem 1. *As $\lambda \to 0$, the first m elements of the solution $\tilde{\mathbf{x}}$ to $\tilde{\mathbf{z}} = \tilde{\mathbf{H}}\tilde{\mathbf{x}}$ are equivalent to the solution $\mathbf{x}$ to Eq. (2.5) - the original SE problem.*

Proof. We split $\tilde{\mathbf{z}} = \tilde{\mathbf{H}}\tilde{\mathbf{x}}$ into the following two equations:

$$\mathbf{0}_m = \mathbf{x} + \hat{\mathbf{H}}^T\mathbf{y}; \tag{4.3}$$

$$\hat{\mathbf{z}} = \hat{\mathbf{H}}\mathbf{x} - \lambda\mathbf{y}. \tag{4.4}$$

Then, by combining Eqs. (4.3) and (4.4), we have:

$$\mathbf{x} = \left(\hat{\mathbf{H}}^T\hat{\mathbf{H}} + \lambda\mathbf{I}_{m\times m}\right)^{-1}\hat{\mathbf{H}}^T\hat{\mathbf{z}}, \tag{4.5}$$

which is the first m elements of $\tilde{\mathbf{x}}$. When $\lambda \to 0$, we then have Eq. (4.1), which is exactly the solution to Eq. (2.5). □

Given Theorem 1, we can translate the original SE problem, whose goal is to solve $\tilde{\mathbf{z}} = \tilde{\mathbf{H}}\tilde{\mathbf{x}}$ deterministically, to an equivalent probabilistic inference problem on a PGM. To be specific, we first convert $\tilde{\mathbf{z}} = \tilde{\mathbf{H}}\tilde{\mathbf{x}}$ to an optimization problem as follows:

$$\tilde{\mathbf{H}}\tilde{\mathbf{x}} = \tilde{\mathbf{z}} \Longleftrightarrow \tilde{\mathbf{H}}\tilde{\mathbf{x}} - \tilde{\mathbf{z}} = 0 \tag{4.6}$$

$$\Longleftrightarrow \min_{\tilde{\mathbf{x}}} \big(\frac{1}{2}\tilde{\mathbf{x}}^T\tilde{\mathbf{H}}\tilde{\mathbf{x}} - \tilde{\mathbf{z}}^T\tilde{\mathbf{x}}\big) \tag{4.7}$$

$$\Longleftrightarrow \max_{\tilde{\mathbf{x}}} \big(-\frac{1}{2}\tilde{\mathbf{x}}\tilde{\mathbf{H}}\tilde{\mathbf{x}} + \tilde{\mathbf{z}}^T\tilde{\mathbf{x}}\big), \tag{4.8}$$

where the equivalence between Eq. (4.7) and Eq. (4.6) is obtained by setting the derivative of the term in Eq. (4.7) with respect to $\tilde{\mathbf{x}}$ to be zero. Then, we can obtain the solution to Eq. (2.5) via maximizing the following joint Gaussian probability density function (pdf):

$$p(\tilde{\mathbf{x}}) = \exp\big(-\frac{1}{2}\tilde{\mathbf{x}}^T\tilde{\mathbf{H}}\tilde{\mathbf{x}} + \tilde{\mathbf{z}}^T\tilde{\mathbf{x}}\big). \tag{4.9}$$

We now can use a Markov network to characterize the joint Guassian pdf. Specifically, we begin by constructing an undirected graph $\tilde{G} = (\tilde{V}, \tilde{E})$, where $\tilde{V}$ is a set of nodes in one-to-one correspondence with the augmented state variables $\tilde{\mathbf{x}} = [\mathbf{x}; \mathbf{y}]$, and $\tilde{E}$ is a set of undirected edges determined by the non-zero entries of the square matrix $\tilde{\mathbf{H}}$. Then, we factorize the joint Gaussian pdf into the following form:

$$p(\tilde{\mathbf{x}}) = \frac{1}{Z}\prod_{\tilde{v}_i \in \tilde{V}} \phi_{\tilde{v}_i}(\tilde{x}_i) \prod_{(\tilde{v}_i, \tilde{v}_j) \in \tilde{E}} \psi_{\tilde{v}_i, \tilde{v}_j}(\tilde{x}_i, \tilde{x}_j), \tag{4.10}$$

where

$$Z = \sum_{\tilde{x}_i} \prod_{\tilde{v}_i \in \tilde{V}} \phi_{\tilde{v}_i}(\tilde{x}_i) \prod_{(\tilde{v}_i, \tilde{v}_j) \in \tilde{E}} \psi_{\tilde{v}_i, \tilde{v}_j}(\tilde{x}_i, \tilde{x}_j). \tag{4.11}$$

$\phi_{\tilde{v}_i}$ and $\psi_{\tilde{v}_i, \tilde{v}_j}$ are the node potential and edge potential associated with the undirected graph $\tilde{G}$, which are defined as:

- **node potential**: $\phi_{\tilde{v}_i}(\tilde{x}_i) = \exp\big(-\frac{1}{2}\tilde{H}_{ii}\tilde{x}_i^2 + \tilde{z}_i\tilde{x}_i\big)$;

- **edge potential**: $\psi_{\tilde{v}_i,\tilde{v}_j}(\tilde{x}_i,\tilde{x}_j) = \exp\big(-\frac{1}{2}\tilde{x}_i\tilde{H}_{ij}\tilde{x}_j\big)$.

With the above analysis, we have the following theorem.

Theorem 2. *The solution* $\tilde{\mathbf{x}}$ *to* $\tilde{\mathbf{z}} = \tilde{\mathbf{H}}\tilde{\mathbf{x}}$ *is equal to the inference of the vector of marginal means* μ *over the graph* $\tilde{G}$ *with the associated joint Gaussian pdf* $p(\tilde{\mathbf{x}}) \sim \mathcal{N}(\mu, \tilde{\mathbf{H}}^{-1})$.

The proof of Theorem 2 is provided in [3]. Now, with Theorem 1 and Theorem 2, we have completed the transformation from the original DC SE problem to the problem of probabilistic inference on a PGM.

4.2.2 Transforming AC SE To Inference Problem On A PGM

The goal of AC SE is to solve Eq. (2.4) to produce an estimation of state variables $\mathbf{x}$. However, since $\mathbf{h}(\mathbf{x})$ is a nonlinear function of $\mathbf{x}$, we cannot obtain an analytic solution for $\mathbf{h}(\mathbf{x})$ directly, unlike we have done for DC SE. Therefore, in practice, Newton-type methods are often applied to iteratively estimate the state variables. At each iteration k, we need to calculate Eq. (2.8) to update $\Delta\mathbf{x}^{(k)}$. For the ease of description, we rewrite Eq. (2.8) by omitting the iteration index, as follows:

$$\Delta\mathbf{x} = \big(\mathbf{H}^T\Lambda\mathbf{H}\big)^{-1}\mathbf{H}^T\Lambda(\mathbf{z}-\mathbf{h}(\mathbf{x})). \tag{4.12}$$

Comparing Eq. (4.12) with Eq. (2.5), we observe that the only difference is, Eq. (4.12) uses the term $\mathbf{z}-\mathbf{h}(\mathbf{x})$, while Eq. (2.5) uses the term $\mathbf{z}$. Therefore, we can transform the original AC SE problem to the problem of probabilistic inference on a PGM, by simply denoting $\hat{\mathbf{z}} = \Lambda^{1/2}(\mathbf{z}-\mathbf{h}(\mathbf{x}))$ and then following the same steps as in Section 4.2.1 for DC SE.

4.2.3 Example

Now, we continue with the 3-bus example to illustrate how to construct a PGM. In this example, since $m = 2$ and $n = 3$, we introduce three auxiliary state variables: y_1 for transmission line between Bus 1 and Bus 2, y_2 for transmission line between Bus 1 and Bus 0, and y_3 for transmission line between Bus 2 and Bus 0, as shown in Fig. 4.3a. As a result, the augmented state variables are:

$$\tilde{\mathbf{x}} = [\mathbf{x};\mathbf{y}] = [x_1;x_2;y_1;y_2;y_3], \tag{4.13}$$

the augmented meter measurements are:

$$\tilde{\mathbf{z}} = [\mathbf{0}_2; \Lambda^{1/2}\hat{\mathbf{z}}] = [0; 0; 30; 4; 202.5], \tag{4.14}$$

and

$$\begin{aligned}\tilde{\mathbf{H}} &= \begin{bmatrix} \mathbf{I}_{2\times 2} & \hat{\mathbf{H}}^T \\ \hat{\mathbf{H}} & -\lambda \mathbf{I}_{3\times 3} \end{bmatrix} \\ &= \begin{bmatrix} 1 & 0 & 250 & 250 & 0 \\ 0 & 1 & -250 & 0 & -2000 \\ 250 & -250 & -\lambda & 0 & 0 \\ 250 & 0 & 0 & -\lambda & 0 \\ 0 & -2000 & 0 & 0 & -\lambda \end{bmatrix}. \end{aligned} \tag{4.15}$$

Each auxiliary state variable is represented as an additional node in the PGM, as shown in Fig. 4.3b. Since y_1 connects to and depends on θ_1 and θ_2, node $\tilde{v}_3$ (for y_1) is connected to both $\tilde{v}_1$ (for θ_1) and $\tilde{v}_2$ (for θ_2) in the PGM. Similarly, we connect $\tilde{v}_4$ (for y_2) with $\tilde{v}_1$ (for θ_1), and connect $\tilde{v}_5$ (for y_3) with $\tilde{v}_2$ (for θ_2). Since Bus 0 is the slack bus, it does not have a corresponding node in the PGM.

The node potential of node $\tilde{v}_1$ in the PGM is defined as:

$$\phi_{\tilde{v}_1}(\tilde{x}_1) = \exp\left(-\frac{1}{2}\tilde{H}_{1,1}\tilde{x}_1^2 + \tilde{z}_1\tilde{x}_1\right) = \exp\left(-\frac{1}{2}\theta_1^2\right). \tag{4.16}$$

Similarly, we can obtain the node potentials of other nodes:

$$\phi_{\tilde{v}_2}(\tilde{x}_2) = \exp\big(-\frac{1}{2}\theta_2^2\big), \tag{4.17}$$

$$\phi_{\tilde{v}_3}(\tilde{x}_3) = \exp\big(\frac{\lambda}{2}y_1^2 + 30y_1\big), \tag{4.18}$$

$$\phi_{\tilde{v}_4}(\tilde{x}_4) = \exp\big(\frac{\lambda}{2}y_2^2 + 4y_2\big), \tag{4.19}$$

$$\phi_{\tilde{v}_5}(\tilde{x}_5) = \exp\big(\frac{\lambda}{2}y_3^2 + 202.5y_3\big). \tag{4.20}$$

The edge potential of edge $(\tilde{v}_1, \tilde{v}_3)$ is defined as:

$$\psi_{\tilde{v}_1,\tilde{v}_3}(\tilde{x}_1,\tilde{x}_3) = \exp\left(-\frac{1}{2}\theta_1 y_1 50 h_{11}\right) = \exp\left(-125\theta_1 y_1\right). \tag{4.21}$$

Similarly, we can obtain the edge potentials of other edges:

$$\psi_{\tilde{v}_1,\tilde{v}_4}(\tilde{x}_1,\tilde{x}_4) = \exp\big(-125\theta_1 y_2\big), \tag{4.22}$$

$$\psi_{\tilde{v}_2,\tilde{v}_3}(\tilde{x}_2,\tilde{x}_3) = \exp\big(125\theta_2 y_1\big), \tag{4.23}$$

$$\psi_{\tilde{v}_2,\tilde{v}_5}(\tilde{x}_2,\tilde{x}_5) = \exp\big(1000\theta_2 y_3\big). \tag{4.24}$$

4.3 Solving The PGM Problem Via Gaussian Belief Propagation

The third step is to solve the probabilistic inference problem on the constructed PGM via belief propagation.

4.3.1 Gaussian Belief Propagation

As nodes in our PGM are characterized by Gaussian random variables, the associated belief propagation algorithm is also called Gaussian belief propagation (GBP) [3, 16]. As we have discussed in Section 3.2, for the graph $\tilde{G}$ composed of node potentials $\phi_{\tilde{v}_i}(\tilde{x}_i)$ and edge potentials $\psi_{\tilde{v}_i,\tilde{v}_j}(\tilde{x}_i,\tilde{x}_j)$, we have the *sum-product* rule and *product* rule. With these two rules and the property that the product of Gaussian distributions is still a Gaussian distribution, we have the exact form for each passing message as follows [16]:

$$m_{\tilde{v}_i\to\tilde{v}_j}(\tilde{x}_j) = \mathcal{N}(\mu_{ij}, P_{ij}^{-1}), \tag{4.25}$$

where

$$P_{i\backslash j} = P_{ii} + \sum_{\tilde{v}_k \in N(\tilde{v}_i)\backslash \tilde{v}_j} P_{ki}; \tag{4.26}$$

$$\mu_{i\backslash j} = P_{i\backslash j}^{-1}(P_{ii}\mu_{ii} + \sum_{\tilde{v}_k \in N(\tilde{v}_i)\backslash \tilde{v}_j} P_{ki}\mu_{ki}; \tag{4.27}$$

$$P_{ij} = -\tilde{H}_{ij}^2 P_{i\backslash j}^{-1}; \tag{4.28}$$

$$\mu_{ij} = -P_{ij}^{-1}\tilde{H}_{ij}\mu_{i\backslash j}. \tag{4.29}$$

After several iterations of message passing, i.e., calculating P_{ij} and μ_{ij} until they converge with respect to a small ε, we obtain the marginals of $\tilde{x}_i$'s which are Gaussian random variables $\mathcal{N}(\mu_i, P_i^{-1})$ with precision and mean as follows:

$$P_i^{-1} = (P_{ii} + \sum_{\tilde{v}_k \in N(\tilde{v}_i)} P_{ki})^{-1}; \tag{4.30}$$

$$\mu_i = P_i^{-1}(P_{ii}\mu_{ii} + \sum_{\tilde{v}_k \in N(\tilde{v}_i)} P_{ki}\mu_{ki}). \tag{4.31}$$

Finally, we obtain the solution to Eq. (2.5) as follows:

$$\mathbf{x} = \tilde{\mathbf{x}}[1:m] = [\mu_1; \mu_2; \cdots; \mu_m]. \tag{4.32}$$

4.3.2 Speeding Up Message-Passing In GBP

Recently, it has been showed in [17, 18] that, when calculating the message $m_{\tilde{v}_i \to \tilde{v}_j}(\tilde{x}_j)$ from node $\tilde{v}_i$ to $\tilde{v}_j$, including the message $m_{\tilde{v}_j \to \tilde{v}_i}(\tilde{x}_i)$ from node $\tilde{v}_j$ to $\tilde{v}_i$ can speed up the computation as the passing

messages can be computed in parallel, while not sacrificing the performance. In doing so, we have:

$$\hat{P}_i = P_{ii} + \sum_{\tilde{v}_k \in N(\tilde{v}_i)} P_{ki}; \tag{4.33}$$

$$\hat{\mu}_i = \hat{P}_i^{-1}(P_{ii}\mu_{ii} + \sum_{\tilde{v}_k \in N(\tilde{v}_i)} P_{ki}\mu_{ki}); \tag{4.34}$$

$$P_{ij} = -\tilde{H}_{ij}^2 \hat{P}_i^{-1}; \tag{4.35}$$

$$\mu_{ij} = -P_{ij}^{-1}\tilde{H}_{ij}\hat{\mu}_i. \tag{4.36}$$

Then, we have the marginal Gaussian pdf $\mathcal{N}(\mu_i, P_i^{-1})$ for each $\tilde{x}_i$ with precision and mean as follows:

$$P_i^{-1} = \hat{P}_i^{-1}; \tag{4.37}$$

$$\mu_i = P_i^{-1}\hat{\mu}_i. \tag{4.38}$$

The detailed implementations of our PGM/GBP solver for DC SE and AC SE are shown in Algorithm 1 and Algorithm 2, respectively.

Algorithm 1 GBP_DC (PGM/GBP solver for DC SE)

Input: $G = (V, E)$, $\mathbf{H}$, $\mathbf{z}$, Λ, λ, ε, and T.
Output: $\tilde{\mathbf{x}}[1:m]$.

Initialize $t = 0$ and $\mathbf{x}^{(t)}$ with $x_i^{(t)} \sim \mathcal{N}(0,1)$.
Initialize $P_{ii}^{(t)} = \tilde{H}_{ii}$, $\mu_{ii}^{(t)} = \tilde{z}_i/\tilde{H}_{ii}$; $P_{ki}^{(t)} = 0, \mu_{ki}^{(t)} = 0, \forall k \neq i$.
Initialize $P_{ij}^{(t+1)} = \infty$, $\mu_{ij}^{(t+1)} = \infty$, for $(i,j) \in \tilde{E}$.
Initialize $P_{ij}^{(t+1)} = 0$, $\mu_{ij}^{(t+1)} = 0$, for $(i,j) \notin \tilde{E}$.
Construct matrix $\hat{\mathbf{H}} = \Lambda^{1/2}\mathbf{H}$ and $\hat{\mathbf{z}} = \Lambda^{1/2}\mathbf{z}$.
Construct augmented matrix $\tilde{\mathbf{H}} = \left[\mathbf{I}_{m\times m}, \hat{\mathbf{H}}^T; \hat{\mathbf{H}}, -\lambda\mathbf{I}_{n\times n}\right]$.
Construct augmented graph $\tilde{G} = (\tilde{V}, \tilde{E})$.
while $\|\mathbf{P}^{(t+1)} - \mathbf{P}^{(t)}\|_\infty \geq \varepsilon$ and $\|\mu^{(\mathbf{t+1})} - \mu^{(\mathbf{t})}\|_\infty \geq \varepsilon$ and $t < T$ **do**
  Update $\hat{P}_i^{(t+1)} = P_{ii}^{(t)} + \sum_{\tilde{v}_k \in N(\tilde{v}_i)} P_{ki}^{(t)}, \forall i \in \tilde{V}$;
  Update $\hat{\mu}_i^{(t+1)} = (\hat{P}_i^{(t+1)})^{-1}(P_{ii}^{(t)}\mu_{ii}^{(t)} + \sum_{\tilde{v}_k \in N(\tilde{v}_i)} P_{ki}^{(t)}\mu_{ki}^{(t)}), \forall i \in \tilde{V}$;
  Update $P_{ij}^{(t+1)} = -\tilde{H}_{ij}^{-2}/\hat{P}_i^{(t+1)}, \forall (i,j) \in \tilde{E}$;
  Update $\mu_{ij}^{(t+1)} = -(P_{ij}^{(t+1)})^{-1}\tilde{H}_{ij}\hat{\mu}_i^{(t+1)}, \forall (i,j) \in \tilde{E}$;
  $t = t + 1$.
end while
return $\tilde{x}_i = \mu_i^{(t)} = \hat{\mu}_i^{(t)}/\hat{P}_i^{(t)}, \forall i$.

Algorithm 2 GBP_AC (PGM/GBP solver for AC SE)

Input: $G=(V,E)$, $\mathbf{z}$, Λ, λ, ε, and T.
Output: $\tilde{\mathbf{x}}[1:m]$.
  Initialize $t=0$ and $\mathbf{x}^{(t)}$ with $x_i^{(t)} \sim \mathcal{N}(0,1)$.
  Initialize $\mathbf{x}^{(t)}$ with $x_i^{(t)} = \infty$.
  Compute $\mathbf{h}(\mathbf{x}^{(t)})$ and $\mathbf{H}(\mathbf{x}^{(t)})$.
  Compute $\mathbf{z}^{(t)} = \mathbf{z} - \mathbf{h}(\mathbf{x}^{(t)})$.
  while $\|\mathbf{x}^{(t+1)} - \mathbf{x}^{(t)}\|_\infty \geq \varepsilon$ and $t < T$ **do**
    Compute $\Delta\mathbf{x}^{(t)} = \text{GBP\_DC}(G, \mathbf{H}(\mathbf{x}^{(t)}), \mathbf{z}^{(t)}, \Lambda, \lambda, \varepsilon, T)$
    Update $\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} + \Delta\mathbf{x}^{(t)}$.
    Update $\mathbf{h}(\mathbf{x}^{(t+1)})$ and $\mathbf{H}(\mathbf{x}^{(t+1)})$.
    Update $\mathbf{z}^{(t+1)} = \mathbf{z} - \mathbf{h}(\mathbf{x}^{(t+1)})$.
    $t = t+1$.
  end while
  return $\tilde{\mathbf{x}} = \mathbf{x}^{(t)}$.

4.3.3 Example

We continue with the 3-bus example. Given the node potentials and edge potentials defined in Section 4.2.3, we can define the joint Gaussian pdf in Eq. (4.10) and use GBP to infer the marginal mean for each $\tilde{x}_i$. For instance, for DC SE, by running Algorithm 1 with $\lambda = 1e-5$ and $\varepsilon = 1e-5$, our GBP terminates in three iterations. The estimated mean values in the three iterations are:

$$\mu^{(1)} = [0.0564, -0.3291, 0.4923, -0.4930, -0.0621], \tag{4.39}$$

$$\mu^{(2)} = [0.0174, -0.1013, 0.6444, -0.6441, -0.0803], \tag{4.40}$$

$$\mu^{(3)} = [0.0174, -0.1013, 0.6437, -0.6434, -0.0799]. \tag{4.41}$$

By selecting the first two elements of $\mu^{(3)}$, our PGM method yields the same results as those estimated by existing SE solvers (i.e., $\theta_1 = 0.0174, \theta_2 = -0.1013$, as shown in Section 2.3).

4.4 Existing SE Solvers Versus Our Proposed PGM/GBP Solver

Existing SE solvers either directly calculate the matrix inverse (e.g., using the Gaussian elimination algorithm) to obtain a closed-form solution, or leverage iterative methods (such as the Jacobi iterative method and the Gauss-Seidel iterative method) to accelerate the calculation. It has been proved in [3, 17, 18] that the GBP solver for a system of linear equations is identical to the Gaussian elimination algorithm, and the GBP

solver incorporating two-way message passing (i.e., speedup) is identical to the Jacobi iterative method. In Chapter 5, we will use experimental results to demonstrate the equivalence between our proposed PGM/GBP solver and existing SE solvers in terms of state estimation results.

4.5 Complexity Analysis

Our proposed PGM/GBP solver and existing SE solvers all involve the calculation of matrix inverse: Eq. (2.5) for the DC case or Eq. (4.12) for the AC case. Without loss of generality, we only focus on the computational complexity of solving Eq. (2.5) for the DC case. Similar trends can be observed for solving Eq. (4.12) for the AC case.

As shown in Algorithm 1, our method solves Eq. (2.5) in an iterative process. At each iteration, it first calculates each $\hat{P}_i$ and $\hat{\mu}_i$, with each traversing all nodes and all edges in $\tilde{G}$ once. Then, it calculates each P_{ij} and μ_{ij}, which also traverse all edges in $\tilde{G}$ once. Thus, the computational complexity of our method at each iteration is $4|\tilde{E}|+2|\tilde{V}|$. From Eq. (4.2), we know that $|\tilde{E}| = 2|E|+(m+n)$, $|\tilde{V}| = m+n$, and $|E| > (m+n)$. Therefore, the dominating computational complexity of our method at each iteration is $O(|E|)$. With T_1 iterations to reach the stop conditions, our PGM/GBP solver has a dominating computational complexity of $O(T_1 \cdot |E|)$.

Existing SE methods solve the matrix inverse in Eq. (2.5) either directly or in an iterative process. Direct matrix inverse methods, such as the Gaussian elimination algorithm, involve matrix-matrix additions and multiplications, while iterative methods only require matrix-vector additions and multiplications. Moreover, iterative methods can exploit the matrix sparsity to further reduce the computational complexity [19]. Thus, in general, iterative methods are more efficient than direct matrix inverse methods, particularly when the matrix size is large. More specifically, direct matrix inverse methods have a computational complexity of $O(nm^2+nm+m^3)$ to obtain a closed-form solution to Eq. (2.5). Iterative methods, such as the Jacobi iterative methods and the Gauss–Seidel iterative methods, have a computational complexity of $O(T_2 \cdot |E|)$ to solve Eq. (2.5), where T_2 is the number of iterations to reach the same stop conditions as our PGM/GBP solver.

The main drawback of existing SE solvers is that, under certain conditions, the iterative process converges slowly or even cannot converge (i.e., diverge), which leads to a very large T_2. However, as demonstrated empirically in [3, 18], our PGM/GBP solver converges faster and has a relatively small T_1. As a result, it is more efficient than existing SE solvers. We will further validate this claim in Section 5.3 with experimental results. The reasons for the superior efficiency performance of our proposed PGM/GBP solver may be due to the following reasons. First, by transforming the SE problem to a probabilistic inference problem on a PGM, the physical constraints and relations between components in the power grid are considered explicitly in the PGM and during the computation. Second, by utilizing the GBP algorithm to perform probabilistic inference, each node in the PGM takes messages from all of its neighbor nodes at each iteration, thus progressing more toward the final estimates at each iteration hence a smaller number of iterations.

CHAPTER 5. PERFORMANCE EVALUATION: PGM

5.1 Experimental Setup

We have conducted a number of experiments to: (1) demonstrate the equivalence of existing SE solvers and our proposed PGM method in terms of the resultant state estimates (Section 5.2); and (2) compare their efficiency in terms of computational time (Section 5.3).

All the experiments are conducted using MATPOWER 6.0 [5], which is a power flow solver package available for MATLAB that allows one to conduct state estimation on power system test models. The running time reported in Section 5.3 is tested on a 15-core Linux server with an Intel Core i5 2.7 GHz processor for each core and 252.2 GB memory. We wrote our own codes for performing the proposed PGM/GBP solver for AC SE. I personally developed the code for generating the meter measurements, which is documented in Appendix A. The parameters used throughout the experiments include:

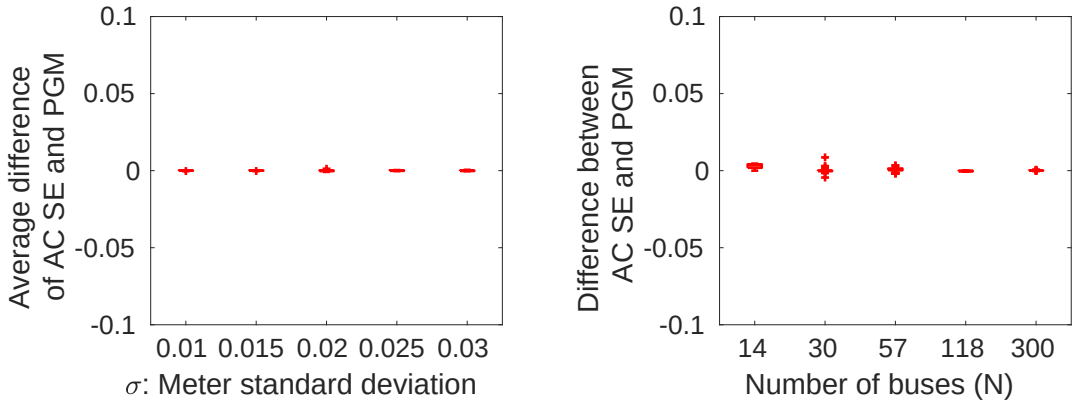
- N: Number of buses in the network;
- σ: Meter standard deviation;
- ε: Threshold value to terminate the iterations;
- T: Maximal number of iterations;

5.2 AC SE Versus AC PGM: State Estimates

We first compare AC PGM and AC SE in terms of the estimated states. The first experiment is evaluated on the IEEE 14-bus system with varying meter standard deviations. Five trials are conducted for each level of meter standard deviation, where different meter measurements are generated according to the corresponding standard deviation. Results are plotted in Fig. 5.1a. The second experiment is evaluated on IEEE 14-bus, 30-bus, 57-bus, 118-bus, and 300-bus systems, with a fixed meter standard deviation of between 0.01 and 0.02, depending on the meter type. The configurations of these test systems can be found in their respective source files provided with MATPOWER as *case14.m*, *case30.m*, *case57.m*, *case118.m*, and *case300.m*,

respectively. Results are plotted in Fig. 5.1b. From these experiments, we observe that both methods produce almost identical state estimates, regardless of the meter accuracy or the power grid size, which demonstrates the practical equivalence between the two methods.

Note that small differences between the state estimates of the two methods have been observed in some of the trials. This is because these two methods have different structures of the iterative process and thus may terminate at different state values. However, since the termination threshold is small (i.e., $\varepsilon = 1e-4$), differences are very minimal.



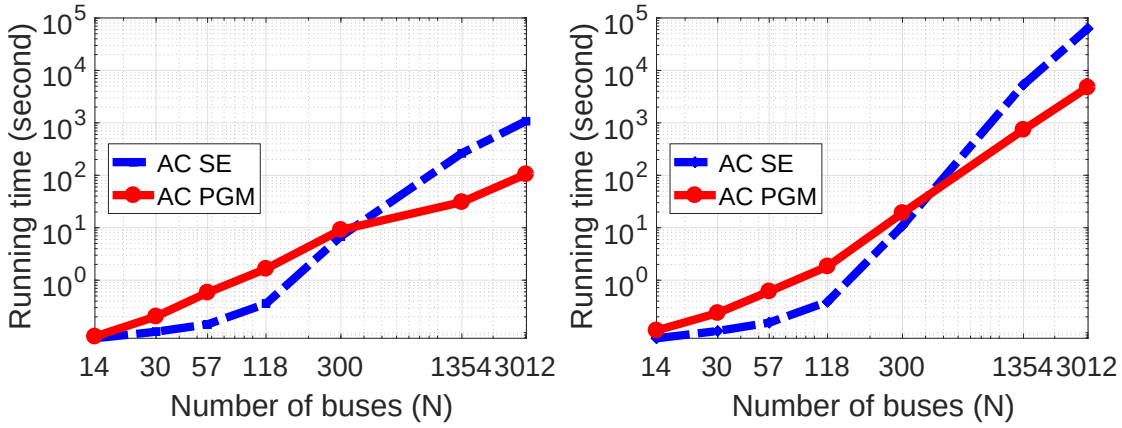
(a) IEEE 14-bus (b) $\sigma = 0.01$ to 0.02

Figure 5.1 Comparison of state estimates for AC SE and AC PGM.

5.3 AC SE Versus AC PGM: Running Time

We compare AC PGM with AC SE in terms of running time with different bus systems and different thresholds ε to compare their computational efficiency. The experiment is evaluated on IEEE 14-bus, 30-bus, 57-bus, 118-bus, 300-bus, as well as Pan European Grid Advanced Simulation and State Estimation (PEGASE) 1354-bus and Polish 3012-bus systems. The configurations of latter two systems also can be found in MATPOWER as *case1354pegase.m* [20, 21] and *case3012wp.m*, respectively. The maximal number of iterations T is set to be 1000. Figs. 5.2a and 5.2b display the running time of the standard AC SE solver and our proposed AC PGM method when $\varepsilon = 1e-4$ and $\varepsilon = 1e-5$, respectively. In each trial,

a variety of real and reactive power flows are measured, and both voltage magnitude $|u|$ and phase θ are estimated.



(a) $\varepsilon = 1e-4$ (b) $\varepsilon = 1e-5$

Figure 5.2 Comparison of running time for AC SE and AC PGM.

We have the following observations: First, when the number of buses (N) is smaller, AC PGM needs slightly more running time. This is because the matrix size dominates the running time when the number of buses is small. Recall from Section 4.5 that AC PGM solves a nonlinear system with matrix size $(m+n) \times (m+n)$, while AC SE deals with a smaller nonlinear system with matrix size $m \times n$. Second, as the number of buses is larger than a certain value, i.e., exceeds 300, AC PGM consumes less time than AC SE. This is because the dominating computational complexities of both methods have the same order: $O(E(\mathbf{H}))$, while AC PGM needs far fewer iterations to converge than AC SE (see Section 4.5). For instance, when $N = 3012$ and $\varepsilon = 1e-5$, AC PGM converges at 98 iterations and 4734 seconds, while AC SE does not even converge within 1000 iterations or 62496 seconds.

To summarize, we have experimentally verified that our proposed PGM solver for AC SE is more computationally efficient, hence more practical, than existing solvers for power networks with a large number of buses.

CHAPTER 6. APPLICATION: DEFENSE AGAINST FALSE DATA INJECTION ATTACKS

With our proposed PGM method as a fast and efficient alternative to solve the SE problem, there are many practical applications for this method. In this chapter, as an example, I explain how our PGM method may provide a practical defense against the well-known false data injection attacks (FDIAs). Table 6.1 summarizes the terms and notations used in Chapter 6.

Table 6.1 Terms and notations used in Chapter 6

Symbol	Definition
FDIA	False data injection attack
BDD	Bad data detection
WSE or J	Weighted squared error
r	Residual
SSR	Sum of squared residuals
σ^2	Meter measurement variance
τ	BDD threshold
α	BDD significance level
$\mathbf{z}_a$	Altered measurement vector
$\mathbf{a}$	Attack vector
$\mathbf{x}_{bad}$	Corrupted state estimates
$\mathbf{c}$	State error injection vector

6.1 State Estimation (SE) And Bad Data Detection (BDD)

After the state variables are estimated, it is important to check whether the estimated state values are trustworthy and whether the measurements are accurate. The bad data detection (BDD) technique is used for this purpose.

BDD is performed using an objective function in the form of a *weighted squared error* (WSE), which is calculated based on the estimated state values, the accuracy of the measuring meters, and the actual meter measurements. The WSE is a weighted *sum of squared residuals* (SSR), where a residual is the difference between an observed measurement and an estimated measurement based on the estimated state values. For

a residual of $r = \mathbf{z} - \mathbf{H}\mathbf{x}$, the SSR is defined as $R = ||\mathbf{z} - \mathbf{H}\mathbf{x}||_2$, where $||\cdot||_2$ is the ℓ_2-norm. The WSE can then be determined from the SSR by taking into account the variances of the meter measurements and by assigning greater weights to the more accurate meters, as follows [22]:

$$J = \sum_{i=1}^{n} \frac{(z_i - \mathbf{H}x_i)^2}{\sigma_i^2}. \tag{6.1}$$

Note that J^2 follows a $\chi^2(n-m)$-distribution. Therefore, if $J > \tau$, we say that the estimated state values are untrustworthy with the probability of a false alarm being α, where τ is the threshold that can be decided using the $\chi^2(n-m)$-distribution at the significance level of α [1].

6.2 False Data Injection Attacks (FDIAs)

In practice, for large-scale power grid systems, DC SE is a popular option to estimate state variables and correct measurement noises, because it is much more efficient than AC SE, which is computationally expensive and thus less feasible.

Unfortunately, DC SE is known to be vulnerable to false data injection attacks (FDIAs). FDIA was first reported in [1] and, since then, has been well-studied in the literature. It works as follows. The attacker alters the meter measurements in such a way that the resulting state estimates are corrupted, while the SSR and hence the WSE calculated by DC SE is the same as they would be without FDIA, thus escaping BDD.

To see this, let $\mathbf{z}_a$ be an altered measurement vector such that $\mathbf{z}_a = \mathbf{z} + \mathbf{a}$, where $\mathbf{z}$ is the original meter measurement vector and $\mathbf{a}$ is an attack vector consisting of meter measurement errors injected by the attacker. Further, let the corrupted state estimates be $\mathbf{x}_{bad} = \mathbf{x} + \mathbf{c}$, where $\mathbf{c}$ is the difference between the estimated state values with and without FDIA. We designate $\mathbf{c}$ as the state error injection vector since it contains the error injected by the attacker into the state estimates. Note that the SSR with FDIA is given as follows:

$$||\mathbf{z}_a - \mathbf{H}\mathbf{x}_{bad}|| = ||\mathbf{z}_a - \mathbf{H}(\mathbf{x} + \mathbf{c})|| = ||\mathbf{z} - \mathbf{H}\mathbf{x} + (\mathbf{a} - \mathbf{H}\mathbf{c})||. \tag{6.2}$$

This tells us that FDIA can evade detection if the attacker injects an attack vector $\mathbf{a}$ such that $\mathbf{a} - \mathbf{H}\mathbf{c} = \mathbf{0}$.

When considering such FDIA against the power grid, it is necessary and helpful to classify different kinds of attacks that a potential attacker could try to launch. [1] lists a number of different FDIA scenarios

Table 6.2 FDIA types. Attacker access to the meters: UL (unlimited) or LIM (limited). Attacker resources: UL (nlimited) or LIM (limited). States under attack: RAND (random) or TGT (targeted). Impact on other states: TGT UC (targeted unconstrained) or TGT CON (targeted constrained).

Scenario	**Access to the Meters**	**Attacker Resources**	**States Under Attack**	**Impact on Other States**
	UL or LIM	UL or LIM	RAND or TGT	UC or CON
1	UL	UL	RAND	
2	UL	UL	TGT	UC
3	UL	UL	TGT	CON
4	UL	LIM	RAND	
5	UL	LIM	TGT	UC
6	UL	LIM	TGT	CON
7	LIM	UL	RAND	
8	LIM	UL	TGT	UC
9	LIM	UL	TGT	CON
10	LIM	LIM	RAND	
11	LIM	LIM	TGT	UC
12	LIM	LIM	TGT	CON

(shown in Table 6.2), which can be classified according to: (1) whether an attacker has unlimited access to all meters or only limited access to some meters in the network; (2) whether an attacker has unlimited resources with which to pollute all of the accessible meters or only limited resources with which to pollute only some of the accessible meters; and (3) whether the attacker injects random state estimates with random errors (a random FDIA) or targets specific states to inject with specific errors (a targeted FDIA).

It should be noted that a random FDIA is not the same as introducing random errors in the meter measurements, which can be easily detected by BDD. What is random in the case of a random FDIA is not the errors introduced into the meter measurements, but the errors injected into the state estimates. That is to say, a random FDIA takes place when the attacker introduces specific errors into specific meter measurement values such that the attack will go undetected by BDD, but does not care which states are compromised or by how much they are compromised as a result.

For targeted FDIA, on the other hand, the attacker designs the FDIA so as to inject specific state estimate errors into specific states. In other words, there are specific states that are targeted by the attacker with the purpose of injecting specific errors into them. Targeted FDIA are distinguished into two sub-

classifications: targeted unconstrained FDIA and targeted constrained FDIA. The difference between these two sub-classifications is that for targeted unconstrained FDIA, the attacker designs the attack to introduce specific errors into the targeted states, but designs the attack such that other non-targeted states may be compromised as well. For a constrained FDIA, however, the attacker designs the attack such that only the targeted states are compromised.

I will consider one particular kind of attack in the discussion that follows- a targeted constrained attack where the attacker is assumed to have unlimited access to meters and unlimited resources. In a targeted attack, the attacker designs the FDIA so as to inject specific state estimate errors into specific states. The attack is said to be constrained in that the attacker designs the attack such that only the targeted states are compromised. I chose to focus on this FDIA scenario because: (1) the attacker is granted the advantage of being able to attack all of the meters; (2) in targeting specific states, the attacker presumably has a specific goal in mind (they want to compromise the power grid in a particular fashion); and (3) by only introducing specific false data into the targeted states, the attack may be less visible than if random errors were injected into multiple states.

To illustrate how an FDIA works, I will continue to use the same example 3-bus power system shown in Fig. 2.1. Recall that I showed in Section 2.3 that, with DC SE, the state variables are estimated as:

$$\mathbf{x} = [\theta_1, \theta_2]^T = [0.0174, -0.1013]^T \text{ radians.} \tag{6.3}$$

With the given meter measurements $\mathbf{z}$, the standard deviation of the meter measurements σ, and the Jacobian $\mathbf{H}$, we can calculate the WSE as:

$$J = \sum_{i=1}^{n} \frac{(z_i - \mathbf{H}x_i)^2}{\sigma_i^2} = 0.2345. \tag{6.4}$$

The χ^2-distribution for this example network (with 3 meter measurements and 2 unknown states) yields a WSE threshold of $\tau = 6.635$ for a significance level of $\alpha = 0.01$. As we would expect, the WSE is well below the threshold when there are only measurement noises but without an FDIA attack.

Now, suppose the attacker intends to inject an error of 0.5 radians into state θ_1. That is, the false data injection vector is $\mathbf{c} = [0.5, 0]^T$, and the intended corrupted state estimates are:

$$\mathbf{x}_{bad} = \mathbf{x} + \mathbf{c} = [0.5174, -0.1013]^T. \tag{6.5}$$

The attack vector needed to alter the meter measurements to generate the polluted state is:

$$\mathbf{a} = \mathbf{Hc} = \begin{bmatrix} 5 & -5 \\ 2.5 & 0 \\ 0 & -4 \end{bmatrix} \begin{bmatrix} 0.5 \\ 0 \end{bmatrix} = \begin{bmatrix} 2.5 \\ 1.25 \\ 0 \end{bmatrix}. \tag{6.6}$$

With this attack vector added to the actual meter measurements, DC SE would yield exactly the same WSE as above (i.e., 0.2345), thus producing the corrupted state estimates $\mathbf{x}_{bad}$ while evading BDD.

6.3 Defense Against FDIAs

Many DC SE based defenses [23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35] have been proposed to deal with FDIAs. However, these defenses usually are hard to implement in practice because they often make very strong assumptions [36]. For instance, most of these defenses assume that certain meter measurements can be completed shielded from the attacker, which may not be realistic. Moreover, it has been shown in [36] that, by slightly modifying the FDIA, DC SE based defenses could be rendered ineffective.

6.4 AC-Based Defense

A more intuitive and effective way to defend against FDIAs is to simply apply AC SE to estimate the unknown states. Before illustrating this, it would be helpful to expand upon the difference between AC and DC SE discussed in Chapter 2. As was stated there, the fact that DC SE has a closed form solution makes it significantly more efficient than the iterative methods used for AC SE. However, the closed form solution also makes DC SE more susceptible to FDIA.

Note that when performing AC SE, meter measurements are taken of both real and reactive power injections to buses and power flows across transmission lines in the network. The nonlinear equations used

to calculate the expected real (P) and reactive (Q) power flows from bus$_i$ to bus$_j$ for AC SE are given below in terms of the state variables ($|u|$ and θ_u):

$$P_{ij} = |u_i|^2 g_{ij} - |u_i||u_j|g_{ij}cos(\theta_i - \theta_j) - |u_i||u_j|b_{ij}sin(\theta_i - \theta_j), \tag{6.7}$$

$$Q_{ij} = -|u_i|^2(b_{ij} + b_i) - |u_i||u_j|g_{ij}sin(\theta_i - \theta_j) + |u_i||u_j|b_{ij}cos(\theta_i - \theta_j), \tag{6.8}$$

where $g_{ij} + jb_{ij}$ is the series admittance of the line and b_i is the shunt susceptance at bus i. For DC SE, the series conductance (g_{ij}) and shunt susceptance (b_i) are neglected, and the voltage magnitudes at all of the buses in the network are assumed to be 1 per-unit or $1pu$. As a result, only real power flows (P) are measured for DC SE and only the voltage phases (θ_u) are estimated. Assuming the voltage phases are small, Equation (6.7) is then approximated by the following linear equation [22]:

$$P_{ij} = b_{ij}(\theta_i - \theta_j). \tag{6.9}$$

All of the functions used to obtain $\mathbf{h}(\mathbf{x})$ for DC SE are of this linear form. Accordingly, when the derivative of Equation (6.9) is taken with respect to the voltage phase at every bus to obtain the Jacobian matrix $\mathbf{H}$ from Equation 2.3, the resultant $\mathbf{H}$ is independent of the voltage states, which is what allows for a closed-form solution in DC SE. For AC SE, however, the Jacobian will be consistent of the derivatives of nonlinear Equations (6.7) and (6.8) with respect to the voltage states. These derivatives will themselves be nonlinear equations, making for a much more computationally complex problem, making FDIAs more difficult to implement.

To illustrate this, consider the 3-bus example. The difference here compared with DC SE is that the real power measurements are defined in terms of their actual non-linear relationships to the unknown states (as opposed to the simplified relationships used for DC). For the purposes of this example (as well as in the experiments conducted in Section 7.3), it is important to note that I am making the same assumption that the

voltage magnitude at every bus is $1pu$ for AC SE as is made for DC SE. That is to say, I am performing SE with a simplified version of AC SE in this example.

The reason I am making this assumption is so that a better comparison can be made between DC and AC SE in terms of the impact that the use of non-linear equations in AC SE makes on the ability to detect FDIA. This example shows that by changing $\mathbf{h}(\mathbf{x})$ from a function of linear equations to non-linear equations, SE is able to detect the FDIA that went undetected in the example in Section 6.2. By implication, standard AC SE, which does not make the assumption that the voltage magnitude of every bus is $1pu$, would likewise be able to detect FDIA.

Applying this simplified version of AC SE to the 3-bus problem with the same attack vector as in the previous section, we will observe a significantly larger WSE (i.e., $J = 13.2042$ as compared with $J = 0.2345$ for DC SE), well exceeding the threshold for BDD ($\tau = 6.635$). As a result, the FDIA can be successfully detected.

The reason why AC SE can successfully defend against FDIA is because it considers the underlying nonlinear dependencies between states and measurements. Thus, the Jacobian matrix in AC SE is a function of the unknown states to be estimated and changes at each iteration. This is unlike DC SE, where the Jacobian matrix is constant over all iterations. As a result, in the case of AC SE with the presence of FDIA, the altered meter measurements not only corrupt the state estimates, but also corrupt the Jacobian that is used to obtain the state estimates. This leads to a large weighted squared error, meaning that the FDIA can be detected by the BDD.

Although the AC SE-based defense is effective against FDIA, it is often considered a less practical solution. This is because AC SE is computationally expensive and time-consuming due to its slow convergence or non-convergence property. Now, given that our proposed PGM solver to the AC SE problem is much more efficient than existing solvers and converges much faster, it becomes more realistic to deploy PGM-based AC SE to defend against FDIA in real power systems.

Before moving forward, it should be noted that the potential vulnerability of even nonlinear AC-based SE to FDIA has been demonstrated by Jin et al. [37], who formulated attacks against AC SE as a convex optimization problem, and solved for attack vectors via semidefinite programming. However, the proposed

attack strategy assumes that an attacker has full knowledge of the power grid system. This means that it would still be difficult for an attacker to launch an effective attack against AC SE (and by implication, AC PGM) using this approach, as full knowledge of all system meter measurements is a strong assumption. A more practical threat model would be one in which an attacker is only allowed to have partial knowledge about the system. Perhaps attacks against AC SE under more practical threat models are still possible, but to our knowledge this has yet to be demonstrated. Nevertheless, providing a means of defense against all possible threats, including those with strong assumptions, would still be ideal. That being said, any further means of defense that are developed for AC SE could theoretically be integrated into AC PGM as well, which has the added benefit of being more efficient.

CHAPTER 7. PERFORMANCE EVALUATION: DEFENSE AGAINST FDIA

7.1 Experimental Setup

I conducted a number of experiments to: (1) examine the susceptibility of DC SE to FDIA under differing conditions (Section 7.2); and (2) illustrate the immunity of AC SE/PGM to FDIAs (Section 7.3). All of these experiments were conducted using MATPOWER 6.0 [5]. Because MATPOWER only offers an open-source code for performing AC SE, I updated the provided code to perform DC SE as well (see Appendix B).

All of the experiments in this chapter were conducted using the IEEE 14-bus system. For the experiments in Section 7.3, I made the same assumption for AC SE/PGM as in the 3-bus example in Section 6.4, namely, that the voltage magnitudes are $1pu$ and the same meter measurements are used for AC as DC SE/PGM. Once again, this is done in order to offer a better comparison of the use of nonlinear versus linear equations in SE/PGM for defending against FDIA.

For convenience, I denote our GBP-based solver on PGM that solves with linear and nonlinear equations as DC PGM and AC PGM, respectively. It should be remembered for the purposes of these experiments that the state estimate values obtained via SE and PGM are practically equivalent (see Sections 4.4 and 5.2). For practical considerations, all of the FDIA in these experiments were *constrained targeted attacks* with either *unlimited resources* or *limited resources* (scenarios 3 and 6 as defined in Table 6.2). My reasons for using *constrained targeted attacks* with *unlimited resources* were explained in Section 6.2. I consider FDIAs with limited resources as well to examine the susceptibility to FDIA even when the attacker is assumed to have fewer resources.

The parameters used throughout these experiments are:

- N: Number of buses in the network;
- J: Weighted squared error (WSE);
- σ: Meter standard deviation;

- ε: Threshold value to terminate the iterations;
- T: Maximal number of iterations;
- M: Number of states under the FDIA attack;
- K: Percentage of the meters that an attacker has resources to compromise;
- $\mathbf{c}_i$: State error injected into state i under FDIA;

By default, I set the number of states under attack to M = 2, the percentage of meters that an attacker has resources to compromise to K = 100%, and the state error injection value to $\mathbf{c}_i = \pi/3$ radians. Moreover, I set the threshold to $\varepsilon = 1e^{-5}$ and the maximal number of iterations to T = 1,000 for both SE and PGM. The BDD threshold for WSE is $\tau = 33.41$, which is the threshold level for a $\chi^2(n-m)$-distribution with $n = 30$ meters and $m = 13$ states at a significance level of $\alpha = 0.01$.

7.2 Susceptibility Of DC SE To FDIA

7.2.1 Successful Attack Ratio When Attacker Has Full Access To All Meters

In order to examine the susceptibility of DC SE to FDIA, I conducted an experiment in which the attacker is assumed to have full access to all meters and desires to attack a specific number of states M ranging from 1 to 13 (note that the total number of unknown voltage phase states in the IEEE 14-bus system is 13). For each M, I randomly generated 1,000 unique attack vectors (or less if the maximum number of unique attack vectors for a given M was less than 1,000) that could be used to launch an attack against that specific number

of states. I then determined what percentage of those attacks could be successfully launched for various amounts of attacker resources (K).

Figure 7.1 shows the results. We observe that as K increases, the attacker has more resources to compromise a greater number of meters, and thus can generate a greater number of successful attack vectors. An interesting finding is that the attacker is able to generate a successful FDIA by polluting just 20% of the meters, and can launch a successful attack against 10 states with 80% of the meters.

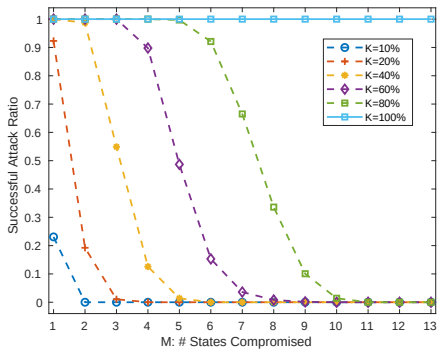


Figure 7.1 Successful attack ratio versus number of compromised states M and amount of resources available K. Y-axis displays the percentage of successfully generated attack vectors with varying amounts of resources. X-axis is the number of states M under attack.

7.2.2 Meters Needed For A Successful Attack

In this experiment, I investigated the percentage of total network meters an attacker needs to pollute to launch a successful FDIA against DC SE that compromises a given number of states, M. The results, shown in Figure 7.2, confirm what we observe in Figure 7.1. For instance, Figure 7.2 shows that between 50 and 70% of the meters would need to be polluted by an attacker in order to successfully launch an FDIA that compromises five state values. Looking at Figure 7.1, we see that an attacker won't be able to attack five states with only enough resources to pollute 40% or fewer of the meters, but can always launch a successful attack against five states with enough resources to pollute 80% of the meters, consistent with the results in Figure 7.2.

As we would expect, the number of meters needed to successfully launch an FDIA generally increases as the number of states being attacked increases. This trend continues until M is so great that all of the available meter measurements must be altered by an attacker (11 or more for the IEEE-14 bus system).

7.3 AC SE/PGM Defense Against FDIA

To evaluate the effectiveness of AC SE/PGM based defenses against FDIAs, I simulated one particular type of strong attacks: a *targeted constrained* attack. Here, the attacker is assumed to have *unlimited*

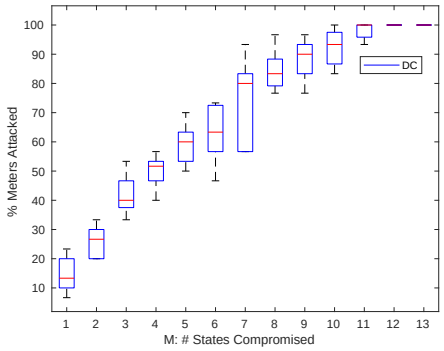
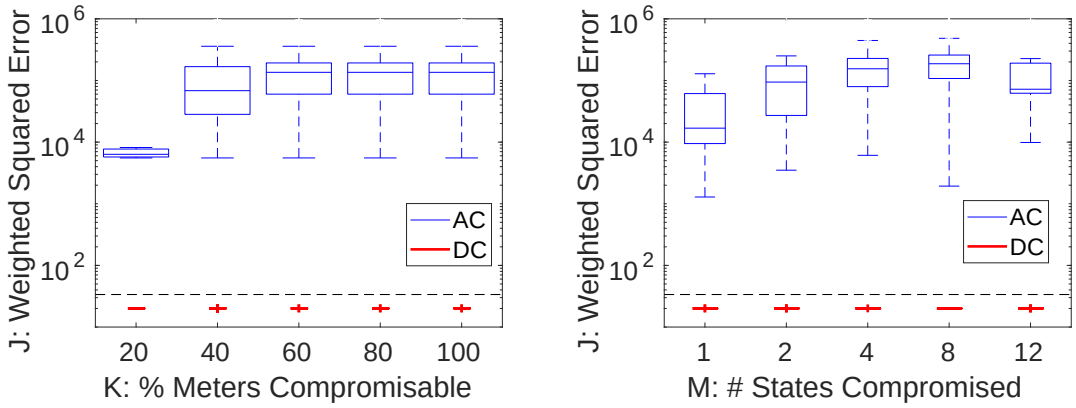


Figure 7.2 Fraction of attacked meters versus number of compromised states M. Y-axis displays the percentage of meter values needed to be changed for a successful attack; X-axis displays the number of states compromised in the attack.

resources and *unlimited* access to the meters. Once again, in a *targeted* attack, the attacker tries to inject specific state estimate errors into specific states; an attack is said to be *constrained* in that the attacker designs the attack in such a way that only the targeted states are corrupted [1].

In these experiments, I show the resilience of AC SE and AC PGM against FDIA. Figures 7.3a and 7.3b display the weighted squared error J for both DC methods (DC SE and DC PGM) and AC methods (AC SE and AC PGM) under successful FDIA against DC SE. For the experiment shown in Figure 7.3a, the attack vectors are generated for various K and $M = 2$. For the experiment shown in Figure 7.3b, the attack vectors are generated for various M and $K = 100\%$. The BDD threshold is shown as the dashed line.

We observe that all attack vectors in both experiments can evade detection by DC methods, as their generated weighted squared errors are below the χ^2 threshold. However, the pollution of meter measurements with these attack vectors can be easily detected by AC methods, whose weighted squared errors are much larger than the threshold. This confirms the resilience of AC SE/PGM against FDIA which are directed at DC SE, as was discussed in in Section 6.4. Taken in concert with the experimental conclusions of Section 5.3, we can conclude that AC PGM is an efficient means of defending against FDIA.



(a) M = 2

(b) K = 100%

Figure 7.3 Comparison of WSE for DC SE/PGM and AC SE/PGM. In (a), we vary the percentage of compromisable meters (denoted as K), and in (b), we vary the number of states to attack (denoted as M).

CHAPTER 8. CONCLUSION

This thesis presents a novel and efficient solver to the state estimation (SE) problem, based on probabilistic graphical models (PGMs). The proposed PGM-based solver (1) models a power grid as a PGM based on the physical constraints of the power grid, (2) transforms the original SE problem into an equivalent probabilistic inference problem on the PGM, (3) uses Gaussian belief propagation (GBP) algorithms to infer the marginal probability distributions of the state variables, and then (4) produces the final state estimates. Experimental results show that the proposed PGM-based solver is more efficient than existing SE solvers, while yielding the same state estimation results. Due to its excellent computational efficiency, our proposed PGM-based solver may have many potential applications. I have highlighted one such application in this thesis, namely, that it can serve as a practical defense against false data injection attacks (FDIAs).

REFERENCES

[1] Yao Liu, Peng Ning, and Michael K Reiter. "False data injection attacks against state estimation in electric power grids". In: *ACM Transactions on Information and System Security (TISSEC)* 14.1 (2011), p. 13.

[2] Gu Chaojun, Panida Jirutitijaroen, and Mehul Motani. "Detecting false data injection attacks in ac state estimation". In: *IEEE Transactions on Smart Grid* 6.5 (2015), pp. 2476–2483.

[3] Danny Bickson. "Gaussian belief propagation: Theory and aplication". In: *arXiv preprint arXiv:0811.2518* (2008).

[4] Daphne Koller, Nir Friedman, and Francis Bach. *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

[5] Ray Daniel Zimmerman, Carlos Edmundo Murillo-Sánchez, and Robert John Thomas. "MATPOWER: Steady-state operations, planning, and analysis tools for power systems research and education". In: *IEEE Transactions on power systems* 26.1 (2011), pp. 12–19.

[6] Allen J Wood, Bruce F Wollenberg, and Gerald B Sheblé. *Power generation, operation, and control*. John Wiley & Sons, 2013.

[7] Jian Sun, Nan-Ning Zheng, and Heung-Yeung Shum. "Stereo matching using belief propagation". In: *IEEE Transactions on Pattern Analysis & Machine Intelligence* 7 (2003), pp. 787–800.

[8] Pedro F Felzenszwalb and Daniel P Huttenlocher. "Efficient belief propagation for early vision". In: *International journal of computer vision* 70.1 (2006), pp. 41–54.

[9] Jeff Bilmes and Geoffrey Zweig. "The graphical models toolkit: An open source software system for speech and time-series processing". In: *2002 IEEE International Conference on Acoustics, Speech, and Signal Processing*. Vol. 4. IEEE. 2002, pp. IV–3916.

[10] Yoshua Bengio. "Markovian models for sequential data". In: *Neural computing surveys* 2.199 (1999), pp. 129–162.

[11] Thomas L Griffiths et al. "Probabilistic models of cognition: Exploring representations and inductive biases". In: *Trends in cognitive sciences* 14.8 (2010), pp. 357–364.

[12] Je-Gun Joung and Zhangjun Fei. "Identification of microRNA regulatory modules in Arabidopsis via a probabilistic graphical model". In: *Bioinformatics* 25.3 (2008), pp. 387–393.

[13] Dror Baron, Shriram Sarvotham, and Richard G Baraniuk. "Bayesian compressive sensing via belief propagation". In: *IEEE Transactions on Signal Processing* 58.1 (2010), pp. 269–280.

[14] Thomas J Richardson and Rüdiger L Urbanke. "The capacity of low-density parity-check codes under message-passing decoding". In: *IEEE Transactions on information theory* 47.2 (2001), pp. 599–618.

[15] Judea Pearl. *Causality*. Cambridge university press, 2009.

[16] Jonathan S Yedidia, William T Freeman, and Yair Weiss. "Understanding belief propagation and its generalizations". In: *Exploring artificial intelligence in the new millennium* 8 (2003), pp. 236–239.

[17] Binghui Wang, Neil Zhenqiang Gong, and Hao Fu. "GANG: Detecting fraudulent users in online social networks via guilt-by-association on directed graphs". In: *2017 IEEE International Conference on Data Mining (ICDM)*. IEEE. 2017, pp. 465–474.

[18] Jinyuan Jia et al. "AttriInfer: Inferring user attributes in online social networks using markov random fields". In: *Proceedings of the 26th International Conference on World Wide Web*. 2017, pp. 1561–1569.

[19] Yousef Saad. *Iterative methods for sparse linear systems*. Vol. 82. siam, 2003.

[20] Cédric Josz et al. "AC power flow data in MATPOWER and QCQP format: iTesla, RTE snapshots, and PEGASE". In: *arXiv preprint arXiv:1603.01533* (2016).

[21] Stéphane Fliscounakis et al. "Contingency ranking with respect to overloads in very large power systems taking into account uncertainty, preventive, and corrective actions". In: *IEEE Transactions on Power Systems* 28.4 (2013), pp. 4909–4917.

[22] Antonio Gomez-Exposito and Ali Abur. *Power system state estimation: theory and implementation*. CRC press, 2004.

[23] Ruilong Deng et al. "False data injection on state estimation in power systems—Attacks, impacts, and defense: A survey". In: *IEEE Transactions on Industrial Informatics* 13.2 (2017), pp. 411–423.

[24] Gaoqi Liang et al. "A review of false data injection attacks against modern power systems". In: *IEEE Transactions on Smart Grid* 8.4 (2017), pp. 1630–1638.

[25] Rakesh B Bobba et al. "Detecting false data injection attacks on dc state estimation". In: *Preprints of the First Workshop on Secure Control Systems, CPSWEEK*. Vol. 2010. 2010.

[26] Aditya Ashok, Manimaran Govindarasu, and Venkataramana Ajjarapu. "Attack-resilient measurement design methodology for state estimation to increase robustness against cyber attacks". In: *2016 IEEE Power and Energy Society General Meeting (PESGM)*. IEEE. 2016, pp. 1–5.

[27] Tung T Kim and H Vincent Poor. "Strategic protection against data injection attacks on power grids". In: *IEEE Transactions on Smart Grid* 2.2 (2011), pp. 326–333.

[28] Oliver Kosut et al. "Malicious data attacks on smart grid state estimation: Attack strategies and countermeasures". In: *Smart Grid Communications (SmartGridComm), 2010 First IEEE International Conference on*. IEEE. 2010, pp. 220–225.

[29] Qingyu Yang et al. “On false data-injection attacks against power system state estimation: Modeling and countermeasures”. In: *IEEE Transactions on Parallel and Distributed Systems* 25.3 (2014), pp. 717–729.

[30] Adnan Anwar, Abdun Naser Mahmood, and Zahir Tari. “Identification of vulnerable node clusters against false data injection attack in an AMI based smart grid”. In: *Information Systems* 53 (2015), pp. 201–212.

[31] Morteza Talebi, Chaoyong Li, and Zhihua Qu. “Enhanced protection against false data injection by dynamically changing information structure of microgrids”. In: *Sensor Array and Multichannel Signal Processing Workshop (SAM), 2012 IEEE 7th*. IEEE. 2012, pp. 393–396.

[32] Shang Li, Yasin Yılmaz, and Xiaodong Wang. “Quickest detection of false data injection attack in wide-area smart grids”. In: *IEEE Transactions on Smart Grid* 6.6 (2015), pp. 2725–2735.

[33] Yi Huang et al. “Defending false data injection attack on smart grid network using adaptive CUSUM test”. In: *Information Sciences and Systems (CISS), 2011 45th Annual Conference on*. IEEE. 2011, pp. 1–6.

[34] Jian Chen and Ali Abur. “Placement of PMUs to enable bad data detection in state estimation”. In: *IEEE Transactions on Power Systems* 21.4 (2006), pp. 1608–1615.

[35] Xuan Liu, Zhiyi Li, and Zuyi Li. “Impacts of bad data on the PMU based line outage detection”. In: *arXiv preprint arXiv:1502.04236* (2015).

[36] Ruilong Deng, Gaoxi Xiao, and Rongxing Lu. “Defending against false data injection attacks on power system state estimation”. In: *IEEE Transactions on Industrial Informatics* 13.1 (2017), pp. 198–207.

[37] Ming Jin, Javad Lavaei, and Karl H Johansson. “Power Grid AC-based State Estimation: Vulnerability Analysis Against Cyber Attacks”. In: *IEEE Transactions on Automatic Control* (2018).

[38] Rui Bo. *test_se*. http://www.pserc.cornell.edu/matpower/docs/ref/matpower5.0/extras/se/test_se.html. [Source code]. 2010 (Accessed April 22, 2019).

[39] Ray Daniel Zimmerman. *runpf*. http://www.pserc.cornell.edu/matpower/docs/ref/matpower5.0/runpf.html. [Source code]. 2014 (Accessed April 22, 2019.

[40] The MathWorks Inc. *normrnd*. https://www.mathworks.com/help/stats/normrnd.html?searchHighlight=normrnd&s_tid=doc_srchtitle. [MATLAB Function]. (Accessed April 22, 2019).

[41] Rui Bo. *doSE*. http://www.pserc.cornell.edu/matpower/docs/ref/matpower5.0/extras/se/doSE.html. [Source code]. 2013 (Accessed April 22, 2019).

APPENDIX A. MATLAB METER MEASUREMENT GENERATION

I developed the following code to generate appropriate meter measurement values that can be used in the state estimation code provided in MATPOWER (function *test_se* [38]). The generated meter measurement values are stored in the measurement index (*idx*), measurement (*measure*), and measurement variance (*sigma*) structs that are defined in *test_se*. To fully integrate my code, lines 17-45 in *test_se*, where these structs are defined, should be replaced with the code included in this appendix. It should be noted, however, that in the version of my code included below the meter measurements are only generated for real power flowing to or from various buses (which are the meter measurements of interest for DC SE). This code can be altered to generate meter measurements for the other parameters on the power grid available for use in AC SE as well.

First, the variance σ^2 of the measurement error is designated in struct *sigma* for each kind of parameter meter as discussed in Section 2.1. The parameters available for measurement in MATPOWER are real power injected into the "from" end of a branch (PF), real power injected into the "to" end of a branch (PT), generator real power injection (PG), reactive power injected into the "from" end of a branch (QF), reactive power injected into the "to" end of a branch (QT), generator reactive power injection (QG), and bus voltage magnitude (Vm). Bus voltage angle (Va) is never measured in SE since it is the state being estimated. In the code included below, we only utilize real power injections to the "from" and "to" ends of a branch in keeping with DC SE. The variance values for PF and PT are the default values provided in the *test_se* function by MATPOWER.

```
% specify measurement variances
sigma.sigma_PF = 0.02;//
sigma.sigma_PT = 0.02; //
sigma.sigma_PG = [];
sigma.sigma_Va = [];
```

```
sigma.sigma_QF = [];
sigma.sigma_QT = [];
sigma.sigma_QG = [];
sigma.sigma_Vm = [];
```

Next, we perform a power flow analysis using the power flow function *runpf* provided with MATPOWER [39]. In this example, this is performed for case14, which corresponds to the IEEE 14-bus model. However, any of the power grid models available in MATPOWER, regardless of the number of buses, could be used instead. The results are stored in struct *results*14 to be accessed later on.

```
% specify measurements

results14 = runpf(case14);
```

The following variables define the index values that will be used to access the power flow results stored in *results*14 for each parameter.

```
PF = 14;
PT = 16;
QF = 15;
QT = 17;
PG = 2;
QG = 3;
VM = 8;
```

Next, the meter measurement values are generated for all possible meters for the parameters of interest (in our case, PF and PT for each branch). This is done by adding a randomly generated value of each meter's error to that meter's per-unit measurement value. The error of each meter is generated by using the MATLAB function *normrnd* [40] to derive a randomly generated number from a normal distribution with mean 0 and variance equivalent to the measurement error variance of each parameter as defined above. The

per-unit measurement values for each meter are simply the real power injections "from" and "to" each branch divided by the base value (100 MVA) as determined by the power flow analysis and stored in *results*14.

```
sizePF = length(results14.branch(:,PF));
MPF = zeros(sizePF,1);
for i = 1:sizePF
  MPF(i) = normrnd(0,sigma.sigma_PF) + (results14.branch(i,PF)/100);
end

sizePT = length(results14.branch(:,PT));
MPT = zeros(sizePT,1);
for i = 1:sizePT
  MPT(i) = normrnd(0,sigma.sigma_PT)+ (results14.branch(i,PT)/100);
end
```

In the following code, the number of states, num_states is defined as 13 (the total number of unknown voltage angles) and the number of meters to be used, num_M, is defined as 30 (though any number greater than 13 and less than the total number of meters could possibly be chosen instead). Out of the total number of meter measurement values that were generated above, 30 are randomly selected. The index for each of these selected meter measurements is also defined.

```
size_M = sizePF + sizePT;

Mvec = zeros(size_M,1);

for i = 1:sizePF
      Mvec(i) = MPF(i);
end
for i = 1:sizePT
      Mvec(i + sizePF) = MPT(i);
end
```

```
num_states = 13;
num_M = 30;

sel_ind = randsample(size_M, num_M);

Mm = sort(sel_ind);

sizePF2 = 0;
sizePT2 = 0;

for i = 1:num_M
   if Mm(i) <= sizePF
      sizePF2 = sizePF2 + 1;
   end
   if Mm(i)>sizePF && Mm(i) <= (sizePT+sizePF)
      sizePT2 = sizePT2 + 1;
   end
end

indPF = zeros(sizePF2,1);
mPF = zeros(sizePF2,1);

for i=1:sizePF2
   indPF(i) = Mm(i);
   mPF(i) = Mvec(Mm(i));
end
indPT = zeros(sizePT2,1);
mPT = zeros(sizePT2,1);

for i=1:sizePT2
   indPT(i) = Mm(i+sizePF2)-sizePF;
```

```
   mPT(i) = Mvec(Mm(i+sizePF2));
end
```

Finally, the meter measurements and their corresponding indices are specified in struct *measure* and struct *idx* to be used in *test_se* when using MATPOWER to perform SE.

```
measure.PF = mPF;
measure.PT = mPT;
measure.PG = [];
measure.Va = [];
measure.QF = [];
measure.QT = [];
measure.QG = [];
measure.Vm = [];

% which measurements are available

idx.idx_zPF = indPF;
idx.idx_zPT = indPT;
idx.idx_zPG = []
idx.idx_zVa = [];
idx.idx_zQF = [];
idx.idx_zQT = [];
idx.idx_zQG = [];
idx.idx_zVm = [];
```

APPENDIX B. MATLAB CODE FOR DC SE

MATPOWER provides MATLAB code within the function *test_se* that can be used to perform AC SE. However, in order to perform DC SE, I had to develop my own code that could be integrated into the code already provided by MATPOWER, specifically within the subfunction *doSE* [41]. Shown below is the code I developed for obtaining the proper values of $\mathbf{h}(\mathbf{x})$ and $\mathbf{H}$ for DC SE as defined in Section 2.1 by replacing the code in *doSE* that is used to update $\mathbf{H}$ every iteration for AC SE. Specifically, lines 82-136 of *doSE*, from the computation of the estimated measurements to the computation of the $\mathbf{H}$ matrix, should be replaced with the code included in this appendix.

As in Appendix A, we only consider the measurements of real power injections into the "from" and "to" ends of bus branches in the code below in keeping with DC SE.

The variable *i* is used in the if statement below to track the number of iterations performed in *doSE* to obtain the state estimates. Since $\mathbf{H}$ is constant for DC SE, it need only be defined during the first iteration. The following section of code details the setup for defining the Jacobian $\mathbf{H}$.

```
%DC SE code

if i == 1

na = size(Va); % Va is a double storing voltage phase states
na = na(1); % na defined as the number of voltage phase states
H = zeros(num_M,n+1); % Jacobian matrix setup with size num_M (number of
    measurements) x n+1 (total number of buses; n is number of unknown
    voltage phases)
sizezPF = size(idx.idx_zPF); % idx.idx_zPF is a double storing the indices
    for real power flow measurements "from" buses
sizezPF = sizezPF(1); % sizezPF stores total number of real power flow
    measurements "from" buses
```

```
sizezPT = size(idx.idx_zPT); % idx.idx_zPT is a double storing the indices
    for real power flow measurements "to" buses
sizezPT = sizezPT(1); % sizezPT stores total number of real power flow
    measurements "to" buses
```

Each of the elements of the Jacobian **H** is defined in the next section of code included on the following page. Recall from Section 6.4 that the elements of the Jacobian for DC SE are derivatives of the linear real power flow equations with respect to the voltage phase state values. In this particular case, each of the elements in the Jacobian **H** is defined in accordance with the following criteria:

- The measurement of each real power injection into a "from" (PF) or "to" (PT) bus corresponds to a row in **H**
- Let the PF measurements correspond to rows $1...n_0$ of **H**, where n_0 is the total number of PF measurements
- Let the PT measurements correspond to rows $n_0+1...n$ of **H**, where n is the total number of PF and PT measurements and $n-n_0$ is the total number of PT measurements
- The voltage phase of each bus (θ_{col}) corresponds to a column in **H**
- Let f designate the "from" bus of the transmission line on which each row's measurement was taken
- Let t designate the "to" bus of the transmission line on which each row's measurement was taken
- For each PF measurement, the linear real power flow equation is $\mathbf{h}_{row} = B_{ft}(\theta_f - \theta_t)$
- For rows $1...n_0$ of **H**, the element in each column is equal to $\frac{\partial \mathbf{h}_{row}}{\partial \theta_{col}}$
- For rows $1...n_0$ of **H**, all elements are zero except in columns f and t
- For rows $1...n_0$ of **H**, the element in column f is B_{ft}
- For rows $1...n_0$ of **H**, the element in column t is -B_{ft}
- For each PT measurement, the linear real power flow equation is $\mathbf{h}_{row} = B_{tf}(\theta_t - \theta_f)$
- For rows $n_0+1...n$ of **H**, all elements are zero except in columns t and f
- For rows $n_0+1...n$ of **H**, the element in column t is B_{tf}

- For rows $n_0 + 1 ... n$ of **H**, the element in column f is -B_{tf}

Note that the transmission line susceptance B_{ft} or B_{tf} is equivalent to the imaginary component of the admittance matrix ($Ybus$) element corresponding to the same "from" and "to" buses. Given the criteria from the previous page, a Jacobian matrix for DC SE can be defined in MATLAB as follows:

```
fr = [];
tr = [];
for ia = 1:sizezPF
   fr = f(idx.idx_zPF(ia));
   tr = t(idx.idx_zPF(ia));
   for l = 1:na
      if fr == l
      H(ia,l) = imag(Ybus(fr,tr));
      elseif tr == l
         H(ia,l) = -1*imag(Ybus(fr,tr));
      end
   end
end

fr = [];
tr = [];
for ia = 1:sizezPT
   fr = f(idx.idx_zPT(ia));
   tr = t(idx.idx_zPT(ia));
   for l = 1:na
      if tr == l
         H(ia+sizezPF,l) = imag(Ybus(tr,fr));
      elseif fr == l
         H(ia+sizezPF,l) = -1*imag(Ybus(tr,fr));
      end
   end
```

```
end
H = H(:,nonref); % Removes the column for the reference bus and places the
    columns of H in the correct order for use in doSE
end
```

In addition to defining the Jacobian matrix, $\mathbf{H}$, we also need to define each of the linear power flow equations, $\mathbf{h}(\mathbf{x})$, in terms of the voltage phase states during each iteration of *doSE*. Recall from Section 2.1 that the goal of SE is to find the best fit for $\mathbf{z} - \mathbf{h}(\mathbf{x}) = \mathbf{e}$. Since the real power flow equations are simplified for DC SE, we need to update $\mathbf{h}(\mathbf{x})$ accordingly. This can be done by following the criteria given below:

- The measurement of each real power injection into a "from" (PF) or "to" (PT) bus corresponds to a row in $\mathbf{h}(\mathbf{x})$
- Let the PF measurements correspond to rows $1...n_0$ of $\mathbf{h}(\mathbf{x})$, where n_0 is the total number of PF measurements
- Let the PT measurements correspond to rows $n_0+1...n$ of $\mathbf{h}(\mathbf{x})$, where n is the total number of PF and PT measurements and $n-n_0$ is the total number of PT measurements
- Let f designate the "from" bus of the transmission line on which each row's measurement was taken
- Let t designate the "to" bus of the transmission line on which each row's measurement was taken
- For each PF measurement, the linear real power flow equation is $\mathbf{h}_{row} = B_{ft}(\theta_f - \theta_t)$
- For rows $1...n_0$ of $\mathbf{h}(\mathbf{x})$, the element in each row is equal to $B_{ft}(\theta_f - \theta_t)$
- For each PT measurement, the linear real power flow equation is $\mathbf{h}_{row} = B_{tf}(\theta_t - \theta_f)$
- For rows $n_0+1...n$ of $\mathbf{h}(\mathbf{x})$, the element in each row is equal to $B_{tf}(\theta_t - \theta_f)$

The corresponding code in for MATPOWER is as follows:

```
na = size(Va); % Va is a double storing voltage phase states
na = na(1); % na defined as the number of voltage phase states
hx = zeros(num_M,1); % Linear real power flow equation double setupt with
    size num_M (number of measurements) x 1
```

```
sizezPF = size(idx.idx_zPF); % idx.idx_zPF is a double storing the indices
    for real power flow measurements "from" buses
sizezPF = sizezPF(1); % sizezPF stores total number of real power flow
    measurements "from" buses
fr = [];
tr = [];
for ia = 1:sizezPF
    fr = f(idx.idx_zPF(ia));
    tr = t(idx.idx_zPF(ia));
    hx(ia) = imag(Ybus(fr,tr))*(Va(fr)-Va(tr));
end

sizezPT = size(idx.idx_zPT); % idx.idx_zPT is a double storing the indices
    for real power flow measurements "to" buses
sizezPT = sizezPT(1); % sizezPT stores total number of real power flow
    measurements "to" buses
fr = [];
tr = [];
for ia = 1:sizezPT
    fr = f(idx.idx_zPT(ia));
    tr = t(idx.idx_zPT(ia));
    hx(ia+sizezPF) = imag(Ybus(tr,fr))*(Va(tr)-Va(fr));
end
  end
z_est = hx; % z_est is the double used to store the values of hx in doSE
```